LONDON metropolitan university

An Agent-based Adaptive Join Algorithm

For Building Data Warehouses

Qicheng Yu

A thesis submitted to the London Metropolitan University in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Computing

December 2012

Abstract

Making better business decisions in an efficient way is the key to succeeding in today's competitive world. Organisations seeking to improve their decision-making process can be overwhelmed by the sheer volume and complexity of data available from their various operational information systems. Many organisations have responded to this challenge by employing data warehousing technologies to make full use of the information in their systems and address real-world business problems.

As organisations move their operation to the Internet to take the advantages of the new technologies, the data warehouse environments for the organisations become more distributed and dynamic. Meanwhile, applications of a data warehouse have evolved from reporting and decision support systems to mission critical decision making systems, which require data warehouses to combine both historical and current data from operational systems. This presents both challenges and opportunities in the designing and developing of new data warehouse systems for supporting decision-making processes which can deliver the right information, to right people, at the right time, interactively and securely.

In typical distributed data warehouse architectures both the logical layer and physical layer of the data warehouse are used to map physical tables in distributed data marts. The physical layer contains historical data materialised in a longer time period while most recent data is only available from the logical layer. To extract knowledge from this data is often expensive, as it usually requires complex queries involving a series of joins and aggregations. Many commercial data warehouse systems place limits on such operations at runtime or sacrifice precision by using approximate replication.

The join operation is one of the most expensive operations in query processing as it combines, compares and merges potentially large data sets. Joining large tables could consume a significant amount of the system resources including CPU, disk, buffer and network bandwidth. Consequently join performance has a considerable impact on overall system performance especially in a distributed data warehouse environment. The traditional 'optimise-then-execute' query processing paradigm is inadequate in this case.

This thesis investigates the evolution of data warehouses to identify architecture suitable for highly distributed data warehouses and studied the feasibility and effectiveness of utilising software agent technology for distributed information systems. A novel agent-based adaptive join algorithm called AJoin for effective and efficient online join operations in distributed data warehouses has been proposed to seamlessly integrate dynamic integration approach with traditional data warehousing technologies to address the issues arising from distributed and dynamic data warehouse environments. Taking into consideration data warehouse features, AJoin utilises intelligent agents for dynamic optimisation and coordination of join processing at run time. Key aspects of the AJoin algorithm have been implemented and evaluated against other modern adaptive join algorithms. The experimental evaluation results demonstrate that AJoin consistently outperforms other adaptive join algorithms under various distributed and dynamic data warehouse environments in this study. The outcome of this research has been very encouraging. The average performance of AJoin in matching the first 50 tuples has improved as much as 67% and overall join performance has improved more than 35% compared with other join algorithms in a distributed and dynamic data warehouse environment.

Acknowledgements

I first wish to thank Dr. Fang Fang Cai for his active supervision during the course of this PhD. Without his key advice for research directions, patient guidance, support and encouragement, this work would not have been possible. He has always demanded the best of me, but also shown great faith in me at times when I was behind due to health problems and was extremely busy in teaching and related activities. I owe to him a lot of what I have learned.

I would also like to thank Dr. Julie A McCann for providing me her original ideas, research direction advice, quality feedback and giving me the opportunity to visit her research group. It has been a great pleasure to work under her guidance.

I would like to give special thanks to Mr. Tariq Bhatti for providing financial support and allowing me to use data from his International Telecommunication System company for the PhD research.

A lot of appreciation and gratitude goes to my colleagues and fellow researchers who have offered me support, inspiration and encouragement during my PhD research. In particular, I would like to thank Preeti Patel for detailed proofreading of several chapters of the thesis.

Finally, I would like to thank my parents, my wife, and my daughter for their encouragement and everyday support. I owe to them every opportunity I had in life, and I am grateful to them for simply being there.

Table of Contents

List of Figures

List of Tables

List of Acronym

S

T

V

W

X

Chapter 1. INTRODUCTION

Making better business decisions in an efficient way is the key to succeeding in today's competitive environment. Organisations seeking to improve their decision-making process can be overwhelmed by the sheer volume and complexity of data available from their various Online Transaction Processing (OLTP) systems. Making this data available to a wide audience of business users is one of the most significant challenges for today's IT industry and professionals.

In response, many organisations choose to employ a data warehouse technology to 'unlock' the information in their OLTP systems and understand real-world business problems. The data warehouse is an integrated store of information collected from other systems, and becomes the foundation for decision support and data analysis (Inmon, 2005). However, as organisations globalise their operations and traditional private networks are replaced by Virtual Private Networks (VPN) which uses the Internet to provide a cost effective solution to connect distributed networks, data warehouse environments for these organisations become more distributed and dynamic. Meanwhile, applications of a data warehouse have evolved from reporting and decision support systems to mission critical decision making systems, which require data warehouses to combine both historical and current data from operational systems. This presents both challenges and opportunities in the designing and developing of new data warehouse systems for supporting decision-making processes which can deliver the right information, to right people, at the right time, interactively and securely.

As integrated enterprise systems become increasingly sophisticated in terms of functionality and environment, there is a need for a novel approach in building complex software systems efficiently and effectively. A promising direction to this is software agents, which are software entities that have an internal goal and acts on behalf of a user. They are suitable for use in wide variety of applications. In particular, they are well suited for applications which involve distributed computation or communication between components, sensing or monitoring of their environment, or autonomous operation. It is believed that software agent technology can be employed advantageously to tackle the problems in distributed and dynamic data warehouse environments.

In traditional data warehouse systems the data processing activities centres around the analysis and mining of different sets of information. This is achieved through complex query processing. The join operation is vital to this query processing as it combines, compares and merges potentially large data sets. Therefore, the join operation of two or more relations is one of the most important operations in database and data warehouse systems. It occurs frequently in relational queries and is one of the most expensive relational operations (Chen et al., 1995). Joining large tables could consume a significant amount of the system resources including CPU, disk, buffer and network bandwidth. Consequently join performance has a considerable impact on overall system performance especially in a distributed warehouse environment. Since the performance of join algorithms varies significantly with their operation environments, making query processing more adaptive and intelligent in distributed and dynamic environments is essential for data warehouses to work effectively and efficiently. Modern pipelined join algorithms have demonstrated their adaptability to dynamic and unpredictable data environments. However, none of those modern join algorithms have been optimised for

data warehouse environments where data is organised in multi-dimensional model and is required to load from distributed and dynamic data sources. Naturally, an adaptive join approach is sought to provide effective and efficient online join algorithm for distributed data warehouses in dynamic environments.

The primary objective of the research is to use intelligent agents in the population, maintenance, and query processing aspects of a data warehouse. In particular, a new join in data warehousing query processing for distributed and dynamic environments is proposed and evaluated.

Aims of the investigation are identified as follows:

- To investigate the development of data warehouses architecture to identify architecture suitable for highly distributed data warehouses.
- To investigate the feasibility and effectiveness of utilising software agent technology to address some specific issues in data warehouses.
- To conduct an experimental study on the performance of modern join algorithm for distributed environment.
- To propose a framework for an adaptive join algorithm using intelligent agents.
- To evaluate the agent-based join approach against current approaches in distributed and dynamic data warehouse environments.

1.1 Main Contributions to knowledge of the Research

- **Literature review**

A comprehensive critical review of the literature documenting previous research on both data warehousing and software agents (Inmon, 1992),(Widom, 1995), (Firestone, 1998), (Samos, 1998), (Honavar et al., 1998), (Theodoratos, 1999), (Rundensteiner, 2000), (Moeller, 2000), (Browning, 2001), (Kimball, 2002), (Anon, 2002), (Inmon, 2005), (Sen, 2005), (Babin, 2008), (Rhodes, 1996), (Huhns, 1998), (Yang, 1998), (Wooldridge, 2000), (Caragea 2001), (Jennings, 2001), (Kushmerick, 2003), (Arcangeli, 2004), (Ahmad, 2008), (Russell, 2009), (Jennings, 2010) are carried out. From the review, the evolution and improvement of data warehouse architectures have been investigated and architectures suitable for highly distributed data warehouses have been identified. With a clear understanding of software agent technologies, it is believed that software agent technology can be employed advantageously to tackle the problems in distributed and dynamic data warehouse environments.

- **Proposed framework for agent-based data warehouse**

Based on the research and investigation, an Agent-based Data Warehouse (ABDW) approach was proposed to tackle the real-time integration problem in distributed and dynamic data warehouse environments. In the proposal, a Multi-Agent System (MAS) platform consisting of several software agencies (group of software agents) forms an ABDW architecture. It is believed that the problem arising from distributed and dynamic

data warehouse environments can be better tackled in the ABDW architecture. However, to achieve this, effective and efficient join operation is identified as the key technique in the ABDW architecture.

- **Construction of data warehouse scenario and simulation environments**

In order to study and evaluate data warehouse and various join approaches effectively, an experimental data warehouse environment based on real world industrial case has been setup. A typical Distributed Data Warehouse (DDW) architecture and multidimensional data model was used as a scenario. Real world telecommunication industrial data containing billing, outgoing and incoming information of each phone call was gathered from International Telecommunication Systems (I.T.S) Limited for the study.

- **Pilot study of modern join algorithms for data warehouse**

Modern pipelined join algorithms as one of the most important techniques in distributed and dynamic data warehouse environments were focused on and investigated. A pilot study was carried out to seek the most effective and efficient online join algorithms. Four modern online ripple join algorithms were implemented in Java and experimented in four types of simulated network environments with various join conditions. Experimental results enhanced our confidence to address the issues arisen from distributed and dynamic data warehouse environments.

- **Proposal and evaluation of an adaptive join algorithm using intelligent agents**

The finding from the pilot study has indicated that modern adaptive join algorithms can effectively address issues arisen from unpredictable network environment, but those algorithms are designed for general join purpose and it is not optimised for data warehouses in a distributed environment. Therefore, an agent-based join algorithm called

AJoin using intelligent agents was proposed to provide effective and efficient online join algorithm for distributed and dynamic data warehouses environments. The main algorithm of AJoin is implemented and evaluated against those modern adaptive join algorithms such as Hash Ripple Join and XJoin. AJoin has exhibited better performance under distributed and dynamic data warehouse environments. The finding of this research was presented at Computation World, November 2009, Athens, Greece and published at the IEEE Digital Library (Qicheng Yu et al., 2009).

1.2 The Structure of the Thesis

The remainder of this thesis is organised as follows:

Chapter 2 reviews the background of data warehouses, discusses characteristics of data warehouses, investigates the development of architectures and data models for data warehousing. The evolution history of data warehouse architecture and issues around data warehouse architecture evolution are highlighted. The important benefits and features of multi-dimensional model compared with third normalised relational data model for data warehouses are emphasised. This chapter also investigates data warehousing research issues in the distributed and dynamic data environment. Three main approaches proposed to manage distributed information: federated control approach (Sheth, 1990), (Hasselbring, 2000) dynamic integration approach (Nica, 1996), and data warehousing approach (Inmon, 1992) are reviewed. Based on recent development of information systems in industrials and businesses, it is concluded that the data warehousing approach is a better choice of information management for supporting decision-making processes

where high-performance query processing and data analysis is critical. The key data warehousing issues of data synchronisation between a data warehouse and its distributed and dynamic data sources to meet the runtime requirements are highlighted. It also presents both challenges and opportunities in designing and developing distributed data warehouse systems in the dynamic and unpredictable data environment.

Chapter 3 investigates the feasibility and effectiveness of utilising software agent technology to address data synchronisation and query performance issues in data warehouses. A new approach called Agent-based Data Warehousing (ABDW) approach, which aims to deal with dynamic and distributed data integration problem more effectively, is proposed. In this chapter, the general concepts of software agent technology including: definitions, types, architectures, agent communication, multi-agent systems, agent-based development methodologies, agent development tools and applications are reviewed. Following the discussion, a detailed ABDW approach is presented. It aims to use software agent to integrate dynamic integration approach and traditional data warehousing approach seamlessly. Join as one of the key operations for the approach is identified for further investigation.

Chapter 4 studies the join algorithms and the factors that affect their performance in distributed data environments. The join algorithms for the relational database systems are carefully reviewed. The strengths and weaknesses of each type of the join algorithm and the factors that affect performances of joins in distributed data environments are highlighted. This chapter also investigates the state of art modern adaptive join algorithms. The main issues of the current adaptive join algorithms are highlighted. A novel agent-based join algorithm called AJoin for the ABDW is proposed. It aims to

utilise intelligent agents to coordinate a ripple hash join and semi-join operations to adapt the change data environment to achieve the best query performance for data warehouses.

Chapter 5 presents an experimental study on modern adaptive join algorithms based on empirical research methods, in particular, four typical types of join algorithms: nested-loop ripple join, Hash Ripple Join, XJoin and XJoin without second stage (XJoin-No2) are focused. This chapter also describes the data warehouse scenario used as a basis to investigate and evaluate various modern join algorithms for data warehouses. In particular, a distributed data warehouse architecture and multidimensional data model for the data warehouse scenario is discussed. Simulated network environments for the experiments of modern adaptive join algorithms are presented. A comparative study on modern adaptive join algorithms is conducted. The need for an effective and efficient join algorithm for current distributed and dynamic data warehouse environments is highlighted.

Chapter 6 presents AJoin framework and algorithm for the proposed agent-based join algorithm for distributed data warehousing. AJoin adopts the principles of the semi-join to transmit only the join attributes rather than all attributes from relations to start a join. In addition, AJoin enhances the conventional semi-join approach by eliminating unqualified tuples at remote sites to minimise the transmission cost. Furthermore, as semi-join requires multiple scans of relations, it may not be as cost effective in a high speed networking environment. To address this problem, AJoin utilise software agents to dynamically switch between full-join and semi-join at runtime based on a cost-benefits analysis involving network speed. AJoin is also able to divide the join task into a series of sub-tasks for parallel processing in a ripple manner similar to other modern adaptive join.

This enabled join results to be produced as soon as join tuples are matched. Join processing continues even if when one of data sources is temporary unavailable. As a result, the AJoin improves join performance significantly at various network conditions. This chapter also discusses the AJoin framework and its algorithm in details, and provides a cost-benefit analysis for its join strategies.

Chapter 7 presents extensive evaluations of AJoin algorithm in its effectiveness and performance. These evaluations are based on a comparative study on AJoin against other modern join algorithms using the following three assessment matrix: network speed, memory use, and join queries. The overall evaluation results show that AJoin has consistently outperformed the other modern join algorithms. In the slower network setup, AJoin performs particularly well and it improves performance against Hash Ripple Join by an average of 29% - 49%.

Chapter 8 concludes this thesis by summarising contributions and results. In closing, suggested areas for further research are presented.

Chapter 2. DATA WAREHOUSING

Making better business decisions in an efficient way is the key to succeeding in today's competitive world. Organisations seeking to improve their decision-making process can be overwhelmed by the sheer volume and complexity of data available from their various operational information systems. Many organisations have responded to this challenge by employing data warehousing technologies to 'unlock' the information in their systems and address real-world business problems.

Data Warehouses became a distinct type of computer database during the late 1980s and early 1990s. They were developed to meet a growing demand for management information and decision-making support that could not be met by operational database systems. In the initial stage, data warehouses were developed by simply copying the database of an operational system to an off-line server where the processing load of analytical query and reporting does not impact on the operational database system's performance.

However, the main objective of a data warehouse is to support management's decision making process rather than to support daily business transactions. A simple copying of the database of an operational system as a data warehouse cannot be satisfied when an organisation attempts to perform strategic analysis using the same database that is used to perform transaction processing. This is because the data serving needs, the supporting technology, the user community as well as the processing characteristics for transaction processing are different from analytical processing.

Data warehouses in this stage of evolution are integrated stores of information collected from the operational systems into subject-oriented data model to form the foundation for decision support and data analysis. There are many types of data warehouses, based on different design methodologies and philosophical approaches. In 1990s, Ralph Kimball and Bill Inmon, two of the pioneers in the field, created and documented the concepts and principles of data warehouses and provided a commonly accepted definition of data warehouse:

> *"A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process".*
> (Inmon, 2005)

As organisations globalise their operations, the networks communications become increasingly important for business. The traditional private networks based on leased lines are very costly. It has recently replaced by VPN which uses the Internet to provide a cost effective solution to connect distributed networks (Moeller, 2000). The data warehouse environments for these organisations become more distributed and dynamic due to the nature of the Internet communication. Meanwhile, applications of data warehouses have evolved from reporting and decision support systems to mission critical decision making systems. This requires data warehouses to combine both historical and current data from operational systems (Browning, 2001), (Inmon, 2008).

The rest of this chapter is structured as follows. Section 2.1 presents characteristics of data warehouses. The main differences between data warehouse system and operational database systems are discussed. Section 2.2 investigates the development of data

warehouse architectures. This includes the evolutionary history of data warehouse architecture and issues around data warehouse architecture evolution are highlighted. Section 2.3 presents dimensional data model for data warehousing. The important benefits and features of dimensional model compared with third normalised relational data model for data warehouses are highlighted. Section 2.4 investigates data warehousing research issues in the distributed and dynamic data environment. Three main approaches proposed to manage distributed information: federated control approach, dynamic integration approach, and data warehousing approach are reviewed. It is concluded that the data warehousing approach is a better choice of information management for supporting decision-making processes where high-performance query processing and data analysis is critical. The key data warehousing issues of data synchronisation between a data warehouse and its distributed and dynamic data sources to meet the runtime requirements are highlighted. It also presents both challenges and opportunities in designing and developing distributed data warehouse systems in the dynamic and unpredictable data environment.

2.1. Data Warehouse Characteristics

The data warehouse is an integrated store of information collected from other systems, and becomes the foundation for decision support and data analysis. It is designed to overcome some of the problems encountered when an organisation attempts to perform strategic analysis using the same database that is used to perform transaction processing. Although there are many types of data warehouses, based on different design methodologies and philosophical approaches, they all have these common characteristics

in that they are subject-oriented, integrated, time-variant, and non-volatile (W. H. Inmon, 2005).

The first characteristic of a data warehouse is that it is subject-oriented. Operational data source across an organisation tend to hold a large amount of data which is organised according to business applications and functions such as fixed line services, prepaid line services, mobile services, and broadband services for a telecommunication company; while the data warehouse is organised around major subjects such as customer, carrier, product and activity. The operational database system is concerned both with database design and transaction process design to avoid data redundancy, ensure integrity and better performance of transactions. The data warehouse system focuses on data modelling and database design exclusively to provide best performance for analytical processes. Transaction process design is not part of the data warehouse environment. The differences between process/function application orientation and subject orientation can also be indicated at differences in the content of data at the detailed level. Operational application-oriented data contains data to satisfy immediate functional/processing requirements that may or may not be of use to the decision support system (DSS) analyst, while data in a data warehouse excludes data that will not be used for DSS processing.

The second most important characteristic of a data warehouse is that data found within a data warehouse is integrated. The data in a data warehouse is presented in a uniform manner. A data warehouse integrates on-line transaction processing (OLTP) data by using consistent naming conventions, measurements, encoding structures, physical attributes, semantics, and so forth. For example, in many organisations, applications can often use similar data in different formats: dates can be stored in Julian, Gregorian or Japan

standard format; Payment data can be described as Dollars, Pounds, or Euros. Item size can be measured in millimetres, centimetres, or inches. Different applications can also use different terms to describe the same type of data. One application can use the term "credit" instead of "deposit" to represent the amount of money deposit in a bank account. When data loaded in the data warehouse, it should be stored in a single, acceptable format agreed to by business analysts, despite variations in the external OLTP sources. This allows data from across the organisation, such as legacy data on mainframes, data in spreadsheets, or even data from the Internet, to be consolidated in the data warehouse, and effectively cross-referenced, giving the DSS analysts a better understanding of the business.

The third basic characteristic of data warehouse is that it is time variant, where all data in the data warehouse is accurate as of some specific moment in time. This basic characteristic of data in the warehouse is very different from data found in the OLTP systems. In the OLTP systems data is represent the current value at any moment in time. For example, an order-entry application always shows the current value of stock inventory; it does not show the value of inventory at some time in the past. Querying the stock inventory amount later may return a different response. However, data stored in a data warehouse is accurate as of some past point in time because data found in the warehouse represented historical information, which is called time variant. The data stored in a data warehouse typically represents data over a long period of time; perhaps up to ten years or more. Operational systems often contain data over a short period of time from the current date back to ninety days earlier because maintaining large volumes of data can affect performance which is the most important issue to address in OLTP applications in order to ensure transaction processing successfully. In effect, the data

warehouse stores snapshots of the operational data generated over a long period of time in a business. It is accurate for a specific moment in time and cannot be changed, which contrasts with an operational system where data is always accurate at the current moment and can be updated when necessary.

The fourth defining characteristic of the data warehouse is that it is non-volatile that the data in a data warehouse will remain unchanged except reconciliation if necessary. In OLTP systems creating, inserting, deleting, and updating are essential functions and done regularly on a record-by-record basis. But the basic manipulation of data that occurs in the data warehouse is much simpler. There are only two kinds of operations that occur in the data warehouse - the initial loading of data, and the access of data. There is no update of data in the data warehouse as a normal part of processing. There are some very powerful consequences of this basic difference between operational database processing and data warehouse processing. At the design level, the update anomaly is not a factor to be concerned in the data warehouse, since update of data is not required. This means that at the physical level of design, the focus can be taken to optimise the access of data, particularly in dealing with the issues of normalisation and physical denormalisation. Another consequence of the simplicity of data warehouse operation is in the underlying technology used to run the data warehouse environment. Having to support record-by-record update in an on-line mode requires the technology to have a very complex foundation underneath a facade of simplicity. The technology-supporting backup and recovery, transaction and data integrity, and the detection and remedy of deadlock are quite complex but unnecessary for data warehouse processing.

Because of these characteristics of a data warehouse, data warehouses are built under a different development methodology than operational database systems. In the next section, the development of data warehouse architecture and associated methodologies for building data warehouses will be investigated.

2.2. Development of Data Warehouse Architecture

A data warehouse can be defined as architecture for delivering information to knowledge workers. It is not a product, but is often instantiated as a collection of products and processes that, working together, form a delivery system for information. Data warehousing then became the key trend in corporate computing in the 1990s. Many methodologies such as SAP methodology, PeopleSoft methodology, Corporate Information Designs methodology, and Creative Data methodology (Sen & Sinha, 2005) have been proposed to simplify the information technology efforts required to support the data warehousing process. This has led to the development of better data warehousing architectures and approaches for building data warehouses in organisations.

A key to successful data warehousing though is to understand that a data warehouse is not just a collection of technologies but an architecture. Data warehouses can be architected in many different ways, depending on the specific needs of a business. A traditional data warehouse architecture that makes up a centralised data warehouse for the entire enterprise could be designed using top-down approach introduced in 1992 by Bill Inmon, (W. H. Inmon, 1992) considered as the father of Data Warehousing. According to Bill Inmon, one of the leading proponents of the top-down approach to data warehouse design, the top-down architecture is represented in Figure 2.1.(W. H. Inmon, 2005)

In the top-down architecture, the process begins with extraction, transformation, and loading (ETL) process which process data from OLTP data sources and outputs it to a centralised Data Staging Area in which data and metadata are loaded into the data warehouse and a centralised metadata repository respectively. Data Marts viewed as small data warehouse focusing on one subject or functional area are then created from the summarised data warehouse and metadata.

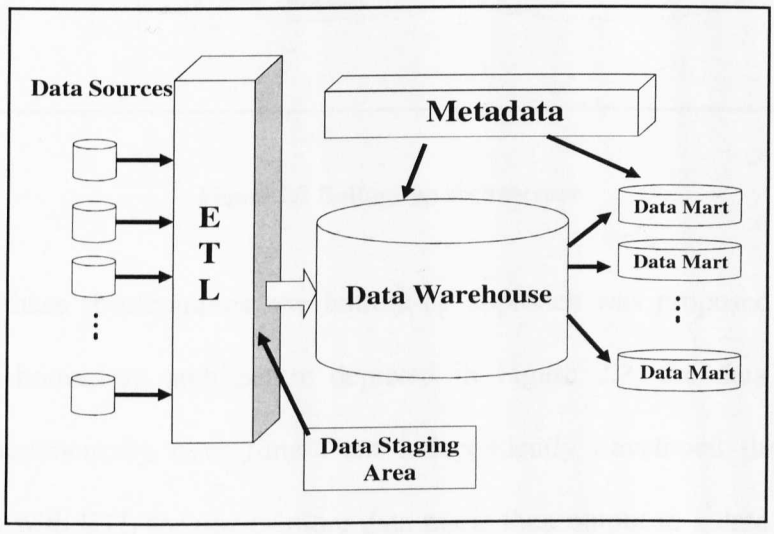

Figure 2.1 Top-down architecture

The data warehouse has a very granular level of data layer and also contains detailed historical data. In contrast, the data marts contain light, highly summarised data and also metadata. The top-down design approach generates highly consistent dimensional views of data across data marts since all data marts are loaded from the centralised repository. Top-down design has also proven to be robust against business changes. The main problems with this architecture are that it usually takes too long to implement and is too expensive to maintain.

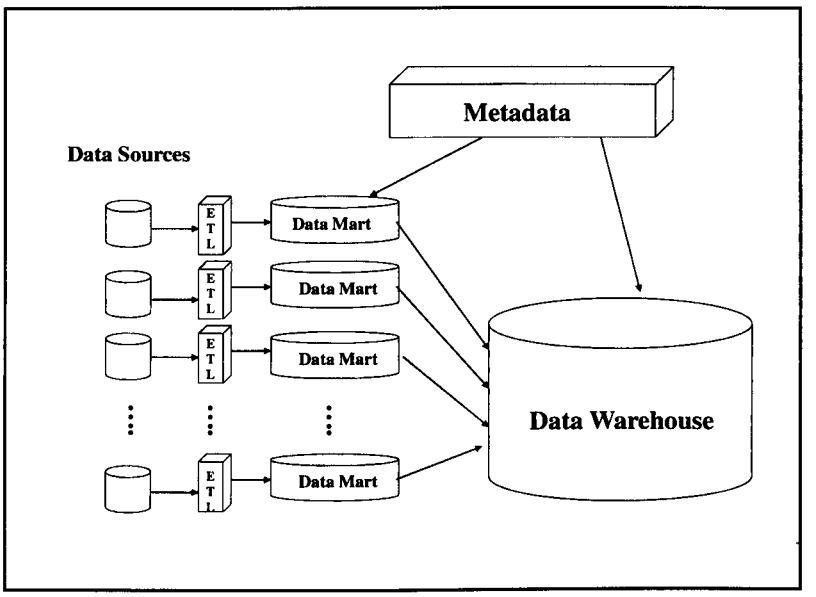

Figure 2.2 Bottom-up architecture

To overcome these shortcomings, the bottom-up approach was proposed by (Hackney, 1997). In the bottom-up architecture depicted in Figure 2.2, the data warehouse is constructed incrementally over time from independently developed data marts. The process begins with ETL for one or more data marts then output to a data warehouse. In the top-down architecture, data marts use lightly and highly summarised data. But in the bottom-up architecture, atomic and detailed historical data is required to store in the data marts, because the data marts become the building foundation of the data warehouse, they must contain all of the data required in the data warehouse. Another important difference between the bottom-up from the top-down architecture is that there is no common metadata components across data marts. The bottom-up architecture was quite successful in meeting initial expectations in building data marts; however, it is often difficult to construct the data warehouse from data marts simply because metadata are not shared.

In response to the challenge, enterprise data mart architecture, also called Bus architecture was proposed by (Kimball et al., 2008). A dynamic data staging area and global metadata repository (GMR) was proposed in the enterprise data mart architecture. In the architecture, depicted in Figure 2.3, no physical organisation-wide data warehouse is implemented. Instead, the data warehouse is viewed as the conjunction of the data marts in the context of a metadata repository. This architecture may allow one to derive aggregate properties that are at the organisation level of analysis, but it will not allow one to derive global properties of the organisation.

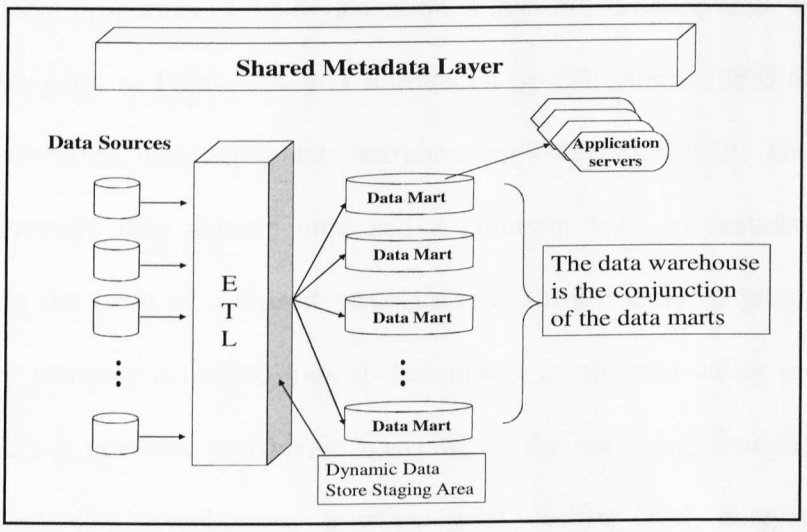

Figure 2.3 Enterprise data mart architecture

Ralph Kimball, a well-known author on data warehousing is often labelled as a proponent of the bottom-up approach to data warehouse design. However, Ralph Kimball said that was misleading, and misunderstandings about his approach. Although the iterative development and deployment techniques may superficially suggest a bottom-up methodology, a closer look reveals a broader enterprise perspective (Kimball et al., 1998).

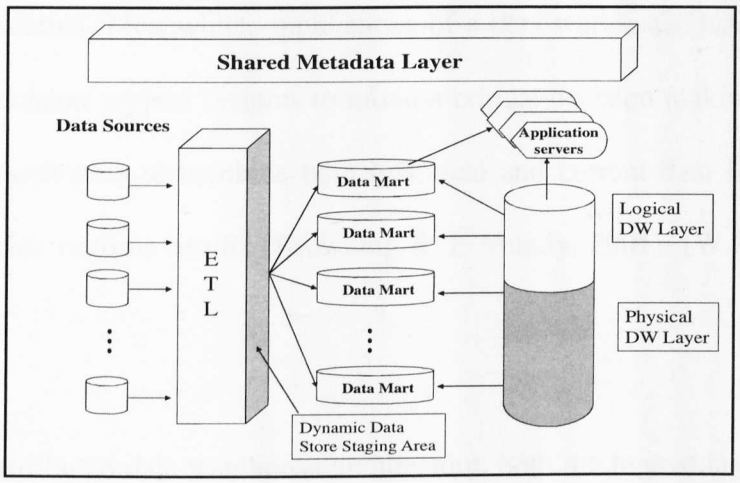

Figure 2.4 Distributed DW/DM architecture

To include global properties of the organisation, a distributed data warehouse / data mart architecture showing in Figure 2.4 was introduced by (Firestone, 1998) and applied in building a web-based distributed data warehousing (Moeller, 2000). The architecture provides a dynamic data staging area and a common view of metadata across the organisation in the form of a shared metadata repository. Firstly, it provides a logical database layer mapping a unified logical data model to physical tables in various data marts. Secondly it provides transparent querying of the unified logical database across data marts and data warehouses, together with caching and integrated services. Consequently, the dynamic characteristics of the data warehouse system become transparent to users. It is a most adaptable architecture, but it still does not support distributed and automated change capture and management which are the most important ability to support real-time decision support systems (DSS).

With the advances in the Internet and web technology, traditional private networks are replaced by VPN which uses the Internet to provide a cost effective solution to connect distributed networks to support organisations global operations. Data warehouse environments for these organisations become more distributed and dynamic due to the

natural of the Internet. Meanwhile, applications of a data warehouse have evolved from reporting and decision support systems to mission critical decision making systems. This requires data warehouses to combine both historical and current data from operational systems to provide runtime results (Browning & J. Mundy, 2001), (W.H. Inmon et al., 2008).

In the typical distributed data warehouse architecture, both the logical layer and physical layer (also called materialised views) of the data warehouse are used to map physical tables in distributed data marts. The physical layer contains historical data materialised in a longer time period such as daily or weekly, while most recent data is only available from the logical layer. Extract knowledge from the most recent data is often expensive, as it usually requires complex queries involving a series of joins and aggregations. Many commercial data warehouse systems (Browning & J. Mundy, 2001), (Lane, 2007) place limits on such operations at runtime or sacrifice precision by using approximate replication proposed by (Olston & Widom, 2005) to overcome the limits. This presents both challenges and opportunities in designing and developing of new data warehouse systems for supporting decision-making processes which can deliver the right information, to right people, at the right time, interactively and securely. We proposed agent-based data warehouse architecture to provide an alternative approach to address the issues. The detailed discussions of the proposed architecture are presented in chapter 3.

2.3. Dimensional Data Model for Data Warehousing

Designing a data warehouse is very different from designing an OLTP system. In contrast to an OLTP system in which the purpose is to capture high rates of data changes and

additions, the purpose of a data warehouse is to organise large amounts of stable data for ease of analysis and retrieval. Because of these differing purposes, there are many considerations in data warehouse design that differ from OLTP database design.

There are two logical design approaches to model data in a data warehouse, namely the dimensional data model and the third normalised form (3NF) data model. In the 3FN model supported by Bill Inmon, the data in the data warehouse are modelled following database normalisation rules. Tables are grouped together by subject areas that reflect general data categories. Inmon suggested that the data model can be constructed with no regard for a distinction between existing operational systems and the data warehouse (W. H. Inmon, 2005). It makes the approach straightforward in design and easy to adapt changes the way in which an organisation does business. However, when applied in large organisations the result is dozens of tables that are linked together by a web of joins. Furthermore, each of the created entities is converted into separate physical tables when the database is implemented (Kimball et al., 1998), (Kimball & Ross, 2002).

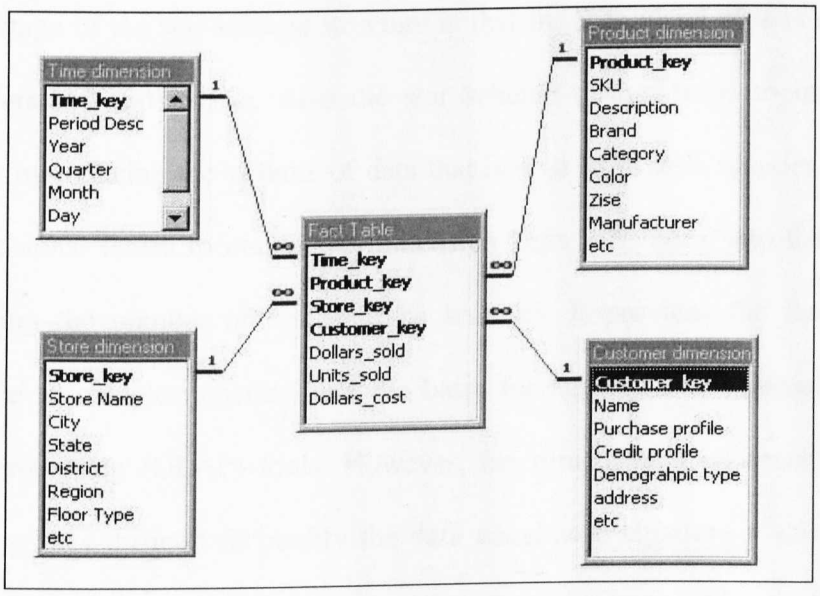

Figure 2.5 A star schema for dimensional model

The dimensional data model is another approach to model data in a data warehouse, in which the data warehouse is modelled using a dimensional model also known as the Star Schema as it resembles a star. Unlike 3NF model using entity-relation modelling that store data in a highly normalised fashion, the data in the dimensional model is stored in vary denormalised manner to improve query performance by reducing join operations at query time. Data warehouses (Joy Mundy et al., 2011) often adopt dimensional modelling that use star and snowflake schemas, depicted in Figure 2.5. A dimension model consists of one fact table and multiple dimension tables. In the centre is a fact table for a fact (measure) of subject area, which can be an event, transaction, or something that happens at a single moment in time. Surrounding the fact table are dimension tables. Each Dimension table contains denormalised contexts of the facts. For example, dimensions related to sales facts can be location including: city, county, state, region, district, etc. or time including: day, week, month, quarter, year, etc.; or product including: description, brand, category, colour, size, etc.

A key advantage of the star schema structure is that the data warehouse is easier for the user to understand and to use. Also the star schema design helps to increase query performance by reducing the volume of data that is read from disk. Queries analyse data in the dimensional tables to obtain the dimension keys that index into the central fact table, reducing the number of tuples to be scanned. It provides the fastest possible response time to complex queries, and the basis for aggregations managed by Online Analytical Processing (OLAP) tools. However, the dimensional approach also has its disadvantages. It is difficult to modify the data warehouse structure if the organisation changes the way of running its business and is complicated to maintain the integrity of

facts and dimensions when loading data into the data warehouse if data is from difference data sources.

It is worth to mention that both approaches can be supported by relation data model and can be presented in entity-relationship diagrams. The difference between the two models is the degree of normalisation. Which model is better, the Dimensional Data Model or the Third Normal Form data model depends on the business cases. For well-established organisations to seeking best query performance to support their decision-making processes, the dimensional data model will be a wise choice. According to (Kimball et al., 2008), the benefits of the dimensional modelling can be summarised into the following:

- Understand ability - Compared to 3NF model the dimensional model is easier to understand and more intuitive. In dimensional models information is organised into coherent business dimensions which make it easier to interpret and understand. It allows users to access a slice of data along any of its dimensions and provides roll-up and drill down functions for efficiently data navigation and analysis. But in 3NF models data is divided into many discrete entities and even a simple business report might require dozens of tables that needs to be joined together in complex way.

- Query performance - Dimensional models are denormalised and optimised for data querying while 3NF models seek to eliminate data redundancies and are optimised for transaction security and performance. In dimensional models, there are standard type of joins and framework. All dimensions can be thought of as symmetrically equal entry points into the fact table. The logical design can be done independent of expected query patterns. The user interfaces are symmetrical, the query strategies

are symmetrical, and the SQL generated against the dimensional model is symmetrical, which allows effectively handle complex queries. Query optimisation for star join databases is simple, predictable, and controllable.

- Extensibility - in a dimensional model, if you need to add some attributes to a dimension, you could just simply add new data rows in the table or executing SQL alter table. No queries or other applications that sits on top of the Warehouse needs to be reprogrammed to accommodate the change. Old queries and applications continue to run without yielding different results. Whereas, a normalised model is more difficult to change under certain conditions. If the change involves adding new data, that, based on normalisation rules, it will be required to create a new entity as well as add foreign keys to whichever other entities are affected.

As dimensional data model has become a widely accepted data modelling approach for data warehouses, understanding the benefits and features of dimensional data model for data warehousing can help us to find the most efficient way to operate data in data warehouses. Since a join operation is one of the most expensive relational operations in dimensional data model, the features of dimensional data model related to join operations such as relationships between a fact table and dimensional tables, cardinality, size of attributes will be further investigated in chapter 5 and 6.

2.4. Approaches for Distributed Information Management

Data warehouses have gained an increasing popularity among organisations which seek to utilise information technology to gain a competitive edge in today's global economy. In

addition to the data warehouse approach, there are other approaches proposed to manage distributed information. The approaches proposed to manage distributed information can be broadly classified into three categories: dynamic integration approach, federated control approach and data warehousing approach.

In the dynamic integration approach (Nica & Elke Angelika Rundensteiner, 1996), users are given direct access to the distributed database schemas at query time by means of special data manipulation language (Litwin et al., 1990), (Bennett & Bayrak, 2011) or metadata resource dictionary (Cheung & Hsu, 1996), (Babin & Cheung, 2008). Because integrated schemas can be promptly created and dropped on the run-time, dynamic integration systems allow the end-user more flexibility and it is easier to maintain in the presence of evolving database schema. Users are able to retrieve the current data available at runtime operational systems. However, there are some disadvantages. They require a more sophisticated end-user who understands the semantics of the schemas in order to access data flexibly. The query performance would be worse if data volumes increase because integration is achieved on the run-time. Moreover, because historical data may not always available at the operational systems, query results are accurate as of the moment of access rather than some point in time which is an important requirement for Online analytical process. The dynamic integration approach alone may not suitable for supporting decision-making processes.

The federated control approach (Sheth & Larson, 1990), also called a Federated Database System (FDBS), is an integrated collection of full-featured distributed databases, in which the component administrator maintains control over their local systems, but cooperate with the federation by supporting global operations (Berthold & Meyer-Wegener, 2001),

(Anon, 2002). The main benefit of the approach is that distributed databases can be well controlled by means of global operations. Therefore it is able to manage distributed information in depth. However, when distributed systems are running autonomously and dynamically, it will be very difficult to provide global operations which can cooperate with autonomous databases. The loosely coupled FDBS are quite similar to the dynamic integration approach and the tightly coupled FDBS are near to a data warehousing approach.

In data warehousing approach (Inmon, 1992), (Kimball et al., 2008), data extracted and integrated from multiple data sources is stored in a centralised repository called data warehouse, which is organised in subjects, containing histories of the data changing over time, and provides with multiple dimensional views. Data is integrated at the time of extracting, and is non-volatile. It could provide users with historical data and facilitate knowledge discovery and on-line analytical processing. Most importantly, because query execution does not involve data integration, complex queries can be executed easily and efficiently. It is a better choice of information management for supporting decision-making processes where high-performance query processing and data analysis is critical. This is particularly advantageous when the information source are very expensive to access or even occasionally become unavailable, when network delays cause high costs, or when the middle-layer task such as transformation and integration are too complex and ineffective, possibly requiring human input.

2.5. Issues in current data warehouse environments

Advances in computer technology and the Internet have revolutionised the way organisations operate their businesses. With higher computer performance and lower storage cost, an increasingly large amount of information are made available in the digital forms and collected and stored in the computer systems. Virtual private networks using the Internet technology to replace the traditional fixed or leased line reduced the cost of telecommunication significantly. It makes easier for organisations to globalise their operations such as opening branches overseas, outsource their call centres in the lower cost foreign countries. New technologies also made more information available from third parties. For example, organisations could access demographic data about customers to improve their customer relationships from the Economic and Social Data Service instead of collecting data by them own. Product price information of competitors could be extracted from competitors' public website.

Due to the development of computer technologies and the changes of the way businesses are running, data sources for an organisation's data warehouse have become more distributed and dynamic. Those data also involves with high volume and often more heterogeneous. As an organisation globalises its operations, the data sources, data marts, and users who needs to access information from the data warehouse of the organisation are distributed worldwide. Data could be managed autonomously at local site to meet their local needs, or data may not be controlled by the organisation if it is retrieved from the third parties. This has made data more dynamic. Dynamic is also reflected on the data communication channels. VNP brings a cost effective solution to connect distributed networks, but it makes the data communication more dynamic and unpredictable due to the dynamic nature of the Internet.

Since a data warehouse has its own separate repository from data sources, information stored in a data warehouse need to be extracted and integrated from multiple data sources in advance. It is hard to keep the data warehouse in synchronisation with various data sources which are often autonomous and dynamic. Although most analysis can be done monthly, weekly or daily, key decisions are increasingly required to be made based on current events to meet the business changes in real-time. Consequently, data warehouses have to be updated continuously. As a result, the query performance advantage of a data warehouse will be seriously affected, and in some cases, it could be even unpractical if data loading processing time longer than real-time query requirements.

Another issue of keeping the warehouse in synchronisation with dynamic data sources is the difficulty of adapting the changes of dynamic data sources. Various information sources could be often autonomous and dynamic as they are generally designed for daily transactions rather than supporting data warehouses. An important consequence of the autonomous data sources is the fact that those sources may change without being controlled from a higher data integration layer. Many sources, particularly web-based third party data sources, may not only change their data and its data structure, but also their capabilities without cooperating with users of their data. Those issues with current data warehouse environment present both challenges and opportunities in designing and developing more adaptive distributed data warehouse systems for supporting decision-making processes in this fast-changing environment.

Some typical issues can be summarised as the following:

- Huge volume and more heterogeneous of data
- Highly distributed data sources

- Dynamic and autonomous data sources
- Dynamic and unpredictable network connections

Chapter 3. Software Agents for Distributed Data Warehouses

Intelligent agent technology is concerned with the development of autonomous computational or physical entities capable of perceiving, reasoning, adapting, learning, cooperating, and delegating in a dynamic environment (Zhong, 2001). It is one of the most promising technologies to design and develop adaptive and distributed systems.

This chapter investigates the feasibility and effectiveness of utilising software agent technology to address some specific issues in data warehouses. A new approach called Agent-based Data Warehousing approach, which aims to deal with dynamic and distributed data integration problem more effectively, is proposed.

The rest of this chapter is structured as follows. In section 3.1, the general concepts of software agent technology including: definitions, types, architectures, agent communication, multi-agent systems, agent-based development methodologies, agent development tools and applications are reviewed. Section 3.2 investigates the feasibility and effectiveness of utilising software agent technology to address some specific issues in data warehouses. Section 3.3 presents a proposed ABDW approach. It aims to use software agent to integrate dynamic integration approach and traditional data warehousing approach seamlessly. Join as one of the key operations for the approach is identified and will be further studied in the following chapters.

3.1. Software Agents

Software agents have evolved from multi-agent systems which in turn form one of three broad areas which fall under Distributed Artificial Intelligence (DAI), the other two being Distributed Problem Solving (DPS) and Parallel AI (PAI). The concept of an agent can be traced back to Carl Hewitt's concurrent Actor model from the early days of research into DAI in the 1970s (Nwana, 1996).

Software agents are software entities which have an internal goal and acts on behalf of a user or other program in a relationship of agency. Software agents are considered one of the most promising technologies to design and develop adaptive and distributed systems as well as to improve on current methods for conceptualizing, designing and implementing software.

3.1.1 Agent Definition

The term 'agent', or software agent, has found its way into a number of technologies and has been widely used, for example, in artificial intelligence, databases, operating systems and computer networks literature. In the rest of this thesis, the term' agent' is synonymous to a 'software agent'. The word agent has been used in a multitude of contexts in computer science. The difficulty in defining an agent is due to the fact that the various aspects of agency are weighted differently based on the application domain. Here are some typical definitions of agents which partly analysed by (S. Franklin & Graesser, 1997):

"Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller."(Smith et al., 1994)

"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."(Maes, 1995)

"Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions."(Hayes-Roth, 1995)

"An agent is a computational entity which:

- *acts on behalf of other entities in an autonomous fashion*
- *performs its actions with some level of proactivity and/or reactiveness*
- *exhibits some level of the key attributes of learning, co-operation and mobility"* (Nwana, 1996).

"Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."(Grosof, 1997)

"Autonomous agents are systems capable of autonomous, purposeful action in the real world." (S. Franklin, 1997)

"An agent is an autonomous software entity that -functioning continuously carries out a set of goal-oriented tasks on behalf of another entity, either human or software system. This software entity is able to perceive its environment through sensors and act upon it through effectors, and in doing so, employ some knowledge or representation of the user's preferences" (N. R. Jennings & M. J. Wooldridge, 1998)

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." (Weiss, 1999)

"Agents are computer systems with two important capabilities. First, they are at least to some extent capable of autonomous action – of deciding for themselves what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents – not simply by exchanging data, but by engaging in analogues of the kind of social activity that we all engage in every day of our lives: cooperation, coordination, negotiation, and the like."(M. Wooldridge, 2002)

"An agent as an entity that receives inputs from its environment evaluates the conditions and performs autonomous actions, perceiving and acting through its own environment to achieve its objectives." (A. Ahmad et al., 2008)

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."(Russell & Norvig, 2009)

Although there is no single definition of an agent, all definitions agree that an agent is essentially special software component which acts on behalf of a user. It works in a relationship of an agency autonomously based on inputs received from its environment through interacting with other agents or its user to achieve its design objectives. An agency usually consists of multiple agents called multi-agent system (MAS). Agents in a MAS may have common or conflicting goals and may interact with each other both indirectly (by acting on the environment) or directly (via communication and negotiation). Agents may decide to cooperate for mutual benefit or may compete to serve their own interests.

Therefore, an agent is autonomous, because it can operate without the direct intervention of humans or others and has control over its actions and internal state. Autonomous agents use their knowledge of their owner's needs and interests to undertake tasks that their owner does repeatedly. The concept of proactiveness is a key element of autonomy. It emphasizes that agents do not simply act in response to their environment. They also exhibit goal-directed behaviour by taking the initiative. (Bollacker et al., 1998), (Rhodes & Starner, 1996) are good examples.

An agent is social, because it cooperates with humans or other agents in order to achieve its tasks. Agents are usually acting in MAS and each agent is given a discrete task, but agents should be able to work in collaboration with other agents, possibly via an agent-communication language to communicate and exchange meaningful messages, to achieve a common goal. Knowledge Query and Manipulation Language (KQML) is the de facto standard agent communication language at present. Agents may share knowledge and learning experiences in the process. This concept is important because a large portion of agent researches (Huhns, 1998), (Lieberman, 1999) are historically rooted in distributed artificial intelligence which emphasizes task decomposition and distribution and collaboration among agents.

An agent is adaptive, because it perceives its environment and responds in a timely fashion to changes that occur in the environment. Agents are able to learn as they react to or interact with their external environment, so that their performance improves over time. The external environment may include the physical world, users (humans), other agents, or the Internet. And an agent is proactive, because it does not simply act in response to its environment but is able to exhibit goal-directed behaviour by taking initiative. (M. Wooldridge & N. Jennings, 1995) pointed out that reactivity and social ability are the two main aspects of the adaptiveness.

Moreover, if necessary, an agent can be mobile, with the ability to travel between different nodes in a computer network. It can be truthful, providing the certainty that it will not deliberately communicate false information. It can be benevolent and always trying to perform what is asked of it. It can be rational, always acting in order to achieve

its goals and never to prevent its goals being achieved, and it can learn, adapting itself to fit its environment and to the desires of its users. It therefore can be considered as an extension to the original concept of autonomy.

Agents are suitable for use in wide variety of applications. In particular, they are well suited for applications which involve distributed computation or communication between components, sensing or monitoring of their environment, or autonomous operation. They are usually continuously running processes that know what to do and when to do it. They communicate with other agents, making requests and performing required tasks. They are also able to migrate in a self-directed way from one host to another on a network.

3.1.2 Agent Types

There are several dimensions to classify existing software agents. They may be usefully classified according to the subset and the importance of features they exhibit, for example, mobile, learning agents. They may be simply classified according to their control architecture, for example, fuzzy agents, and neural network agents. They could be classified according to the tasks they perform, for example, price monitoring agents or email classification agents. They may be also classified according to the environment in which the agent acts, for example, e-commerce agents, telecommunication agents. (S. Franklin & Graesser, 1997)

According to (Nwana, 1996), agents can be classified with respect to the following dimensions:

- Mobility, that differentiates agents into static or mobile
- The logic paradigm they employ, which classifies them as either deliberative or reactive
- The fundamental characteristic that describes the agent (autonomy, cooperatively, learning).

Based on these axes, agents can be classified as collaborative agents, collaborative learning agents, interface agents and truly smart agents shown as Figure 3.1below:

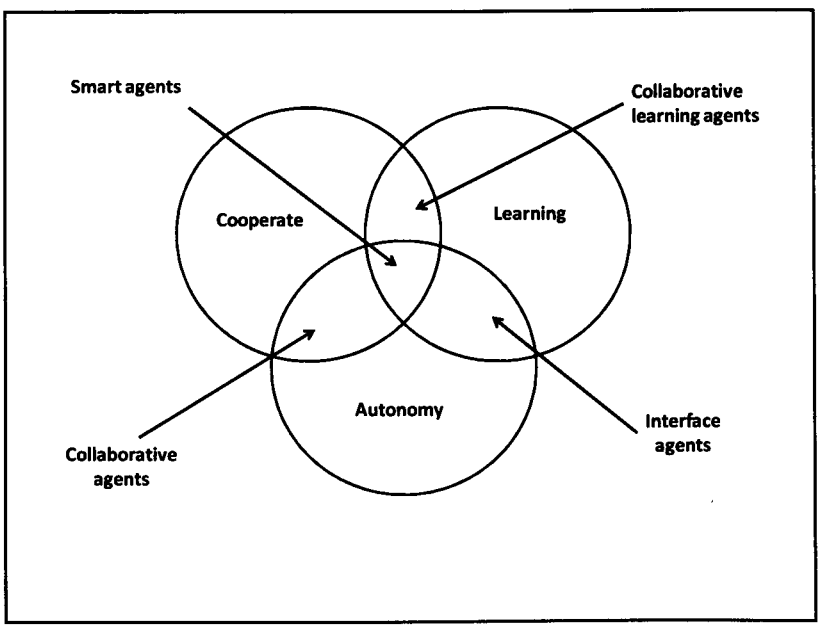

Figure 3.1 The agent classification (Nwana, 1996)

Nwana's classification scheme is robust enough to meet the needs and it covers a wide area of agent-related applications. It is wide accepted by a large number of researchers in the research areas. Based on the above classification, Nwana further simplified them into

six types of agents: collaborative, interface, mobile, information, reactive, hybrid and heterogeneous agents.

- Collaborative agents emphasise autonomy and cooperation with other agents in order to perform tasks for their owners in open and time-constrained multi-agents environments. Each has a degree of expertise about some area and calls upon the expertise of other agents in areas where it lacks knowledge. They may learn, but this aspect is not typically a major emphasis of their operation.
- Interface agents emphasise autonomy and learning in order to perform tasks for their owners. They can act autonomously to perform operations without explicit directions from the user, and, potentially, they can collaborate with user or other types of agents.
- Mobile agents are software processes capable of transporting its state from one environment to another, with its data intact, and capable of performing appropriately in the new environment. For example, mobile agents can to foreign hosts, gathering information on behalf of their owners and coming 'back-home' having performed the duties imposed on them. These duties may range from making a flight reservation to managing a telecommunication network.
- Information agents are agents that have access to at least one, and potentially many information sources, and are able to collate and manipulate information obtained from these sources in order to answer queries requested by their owners and other agents. They perform the role of managing, manipulating or collating information from many distributed resources, such as the Internet.
- Reactive agents represent a special category of agents which do not possess internal, symbolic models of their environments; instead they respond in a stimulus-response

manner to the present state of the environment in which they are embedded. Reactive agents work dates to research such as (Brooks, 1986) and (Agre & Chapman, 1987), but many theories, architectures and languages for these sorts of agents have subsequently been developed.

- Hybrid agents constitute a combination of two or more agent philosophies within a single agent. This alternative is to maximise the strengths and minimise the deficiencies of the most relevant techniques for a particular purpose. In some case, the reactive component would bring robustness, faster response time and adaptability benefits, while the deliberative part would handle the longer term goal-oriented issues.

This research is focused on the distributed data warehousing field. Interface agents, information agents, reactive agents, and mobile agents in MAS are the most important agents, which can be applied to address some of the specific issues arisen from distributed data warehouses.

3.1.3 Agent Architectures

Agent architectures provide the blueprints for the design and development of individual agents. They are the fundamental mechanisms underlying the autonomous agents that support effective behaviour in real-world, dynamic and open environments. Based on our review the agent architectures can be divided into four main groups: logic based, reactive, BDI and hybrid architectures (MÜLLER, 1999), (Nwana, 1996), (Bellifemine et al., 2007), and (N. R. Jennings & M. J. Wooldridge, 2010).

Logic-based architectures are developed from traditional knowledge-based systems techniques in which agents use a symbolic representation of their environment and behaviour and use logical deduction to make decision. The advantage of this approach is easy to understand as human knowledge is symbolic and agents' behaviour can be guaranteed by using logical deduction. The disadvantages are that it is difficult to translate the real world into an accurate, adequate symbolic description, and that symbolic representation and manipulation can take considerable time to execute with results are often available too late to be useful. The architecture is only suitable when environment doesn't change faster than the agent can make decisions.

Reactive architectures implement decision-making as a direct mapping of situation to action and are based on a stimulus–response mechanism triggered by sensor data. Probably the best-known reactive architecture is Brooks's subsumption architecture (Brooks, 1991), in which agents' decision-making is realised through a set of task accomplishing behaviours. Unlike logic-based architectures, they do not have any central symbolic model and therefore do not utilise any complex symbolic reasoning. The advantage of this approach is simple, economic, computationally tractable, and robust against failure. However, the fact that reactive agents do not employ models of their environment results in some disadvantages. In fact, sensor data may not be sufficient to determine an appropriate action and the lack of agent state makes it almost impossible to design agents that learn from experience. Moreover, it is very hard to specific emergent behaviour which involves experimentation, trial and error.

Belief, desire, intention (BDI) architectures are based on four key data structures: beliefs, desires, intentions and plans, and an interpreter (Georgeff & Rao, 1995). They have their roots in philosophy and offer a logical theory which defines the mental attitudes of belief, desire and intention using a modal logic. They are examples of practical reasoning – the process of deciding, moment by moment which action to perform in the furtherance of our goals. Agents not only strive to achieve their goals but must take time to reflect on whether their goals are still valid and possibly revise them and must decide on type of environment to enable a good balance between being committed to intentions and reconsidering them. The advantage of this approach is intuitive as we can recognise decision process and it gives a clear functional decomposition, indicating what sorts of subsystems are required to build an agent. However, how to implement efficiently is still a big challenge.

Hybrid architectures combine two or more types of agent architectures, such as deliberative and reactive architectures. The aim of this architecture is to gain the extra benefits that could not be obtained from a single type. For instance, the reactive component is given more action precedence over the deliberative component to respond rapidly to urgent environmental events. Hybrid agent architectures are usually layered and the layers may be arranged horizontally such as Touring Machines proposed by (Ferguson, 1992) or vertically such as InteRRap suggested by (Müller, 1996).

In horizontal layering, the layers are directly connected to the sensory input and action output and each layer acts like agent, producing suggestions of which action to perform. The main advantage of the horizontal layered approach is the simplicity of design since an agent with n different types of behaviour can be implemented in n different layers.

However, since layers compete with one another to generate action suggestions, their actions could be inconsistent prompting the need for a mediator function to control the actions and the centralised control may also lead to a bottleneck in the agent's decision-making.

In vertical layering, sensory input and actions output dealt with by one only each. The advantage of the vertical layered approach is the complexity of interactions between the layers is reduced since the interaction between layers is reduced significantly. The main disadvantage is the approach is not fault tolerant as control must pass through each different layer and the whole system fails if one layer fails.

3.2. Software Agents for Distributed Information Systems

As software agents have unique properties of autonomy, collaboration, adaptiveness and mobility. They have been widely used for variety of applications particular in the area of distributed information integration and management. In order to further investigate the feasibility and effectiveness of utilising software agent technology in data warehousing, we studied five related software agents: resource agents, mobile agents, intelligent agents, user interface agents and cooperative agents, which are frequently applied to enhance management of information system.

3.2.1 Resource Agents

A resource agent is software agent which exists within a domain to make information resources more intelligent by wrapping them with agent capabilities. The purpose of a resource agent is to mediate access to a particular information resource for other agents; the resource agent understands how to access the resource and also understands the permission structures associated with the resource.

Resource agents have been proposed in Cooperative Information System (CIS) by (Huhns, 1998)as well as a framework for developing mobile agents for managing distributed information resources by (Dale & DeRoure, 1997). In (Huhns, 1998), resource agents are divided in variety of common types, depending on which resources they are representing. The resource agents typically have the following capabilities:

- translating into and from local access languages using wrappers implement common communication protocols
- supporting SQL to manage specific information resources
- utilising machine learning techniques to form logical concepts from data or using statistical techniques to perform data mining
- applying the mappings that relate each information resource to common context to perform a translation of message semantics

In (Dale & DeRoure, 1997) proposal, resource agents are static agents that exist within a domain to provide a level of abstraction between the information resource to which they provide access and the requesting mobile agents. They support the following functions:

- Thoroughly conversant with the protocols of the information resource and understanding how to access the resource and also understands the permission structures associated with the resource.
- Providing an ontological description for each of the services offered by the resource.
- Mediating access to the resource at a resource level. Resource agents can grant access or impose constraints on other agents according to permission requirements.
- Advertising the presence of the information resource by registering the services so that other agents can be aware of what information resources are present.

Extraction information from various information resources is the first important step in data warehouse processing. However, it is challenging to develop and maintain effective data extract functions in a data warehousing system when new and dynamic information sources become available frequently. There is an urgent need for information resources become more 'intelligent' so that the data extract function can adapt the changes of data source easily. The idea of resource agents can be potentially used as new possible solutions to address the issue. In order to make full use of resource agents, an opening standard interface, which can be easily cooperated with MAS platform, need to be established.

3.2.2 Mobile Agents

Mobile agent is one of the important agent types. Mobile agents are capable of transmitting themselves - their program and their state - across a computer network, and

recommencing execution at a remote site as needed for accomplishing their tasks. Mobile agents provide a potentially efficient framework for performing computation in a distributed fashion at sites where the relevant data is available instead of expensive shipping of large volume of data across the network. Mobile agents have been widely used for distributed information management (Dale & DeRoure, 1997), (Honavar et al., 1998), (Yang et al., 1998), (Vieira-Marques et al., 2006).

In Distributed Knowledge Networks (DKN) proposed by (Honavar et al., 1998), the commercially available Voyager mobile agent infrastructure have been used to carry out the computation on site, and return with useful results. The prototype of DKN included representative algorithms from several machine learning paradigms to customise mobile agents for document retrieval from distributed document collections. Similar approaches have been also applied for (Honavar et al., 1998) and (Honavar & Dobbs, 2001) research projects.

In (Dale & DeRoure, 1997)'s research work, mobile agents have been proposed for users to control over their own distributed information resources (through the appropriate resource agents) and gain access to other shared information resources. The essential functions of mobile agents will be defined by the distributed information management tasks that they are allocated. However, they have the following interactions with the framework:

- Determine where to migrate to next by initially querying the domain agent
- Authentication by an electronic signature that they carry
- Running autonomously
- Communication with resource agents as well as other agents

- Transmission the results of their findings and actions regularly to their user

Mobile agents not only can be used to strengthen the effectiveness of data warehouse to gathering information from distributed data sources, but also can be very important to improve the performance of data warehouses accessing especially in Distributed Data Warehouse (DDW) / Distributed Data Mart (DDM) architecture). As mobile agents are foundation of agent-based distributed system, it is a nature idea to enclose mobile agents into agent infrastructure. A standard effective protocol of communication among mobile agents, resource agents, and other mobile agents is essential for mobile agent approach to manage distributed information.

3.2.3 Intelligent Agents

Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it, to goal-directed behaviour by taking the initiative, and to interacting with other agents in order to satisfy their design objectives (M. Wooldridge & N. Jennings, 1995). Intelligent agents offer an attractive approach to manage distributed information since information sources are often autonomously owned and operated, geographically distributed, and dynamic changes.

In (Honavar et al., 1998)'s DKN, intelligent agents were proposed for customised information retrieval, extraction, assimilation, and knowledge discovery. Intelligent agents were applied for monitoring different data sources and routing the appropriate information selectively to relevant sites or specific users. They were also used to detect

and propagate the changes and trigger the necessary updates in the affected data and knowledge repositories.

However, in other agent-based approaches, intelligence has been enclosed in various types of agents rather than adopting particular intelligent agents. For example, in (Huhns, 1998)'s CIS research work, resource agents are capable to apply machine learning techniques to form logical concepts from data or use statistical techniques to perform data mining autonomously.

In our view, reactivity, proactiveness and social ability are basic features of software agents, therefore intelligence should be incorporated in various agents when it requires. For instance, reactivity and proactiveness will be essential functions for resource agents to deal with dynamic, heterogeneous, and distributed information resources. But social ability may be a more important feature for mobile agents when multiple mobile agents are working together. In order to overcome the disadvantages of data warehouse approach for management distributed information to supporting decision-making processes, it will be necessary to enclose intelligent features in various agents applied in data warehouse systems.

3.2.4 User Interface Agents

User interface agents are commonly refers to software agents for human-computer interface which were described as personal assistants who are collaborating with user in

the same work environment (Maes, 1995). User interface agents provide users a window onto their agents, their status, their results and their framework.

In (Dale & DeRoure, 1997)'s research work, a user interface agent has been proposed to:

- Launches mobile agents on behalf of the user and tracks their progress and position
- Provides mobile agents with a communication point through which they can return the results of their tasks
- Organises and pre-processes information returned from mobile agents into a form that is suitable for user
- Provides agent platform with the authentication credentials of user's mobile agents

The user interface agent is designed to generate resource agents automatically base on the interpreted requirements of the user.

In (Huhns, 1998)'s CIS project, user interface agents have following characteristics:

- Contain mechanism to select an ontology
- Support a variety of interchangeable user interfaces
- Support a variety of interchangeable results browsers and visualization tools
- Maintain models of other agents
- Provide access to other information resources

In data warehouse systems, user interface agents could be also very useful. They can be applied to enhance usability and accessibility of the user access interfaces and system

management interfaces. The user interface agents can also used as a connection point for user to cooperate human intelligence with agents' in order to achieve goals.

3.2.4 Cooperative Agents

Cooperative agents make between human and agents, agents and agents communicate each other. They are working together as a group to carry out each task. They will dynamically construct information retrieval plans and integrate the information they extracted for presentation to users. In agent-based distributed information management the individual agents associated with independently managed data sources is often autonomous, the cooperative agents become core mechanisms to coordinate and control over the behaviour of such systems.

In (Honavar et al., 1998)'s DKN project, cooperative agents incorporated coordination and control mechanisms, which were inspired by examples of both natural and artificial systems, into its MAS. The main functions included coordination among agents, synchronisation among multiple agents, activation and deactivation of individual or groups of agents, selection among agents, creation of new agents when needed, elimination of agents that are no longer needed, adaptation of individual agents and group of agents to changes in the environments or task demands, learning from experience, and evolution of agents populations toward more desirable behaviours.

(Honavar et al., 1998) used broker agents and mediator agents to cooperate multiple agents. Broker agents were designed for locating appropriate agents with appropriate

capabilities by means of a "yellow page" and "white pages" directory service. Broker agents also functioned as communication aides by managing communications among various agents, databases, and application programs in the environment. Mediator agents were specialized execution agents, which determined which resources might have relevant information with help from broke agents, decomposed queries to be handled by multiple agents, combined the partial results, and translated between ontologies.

Instead of using cooperative agents, (Dale & DeRoure, 1997) proposed an agent service layer in its agent architecture, which was supported by the transport and control layer, to provide general facilities for managing, naming, controlling and migrating agents. It also provided knowledge-based, goal-directed reasoning. In interchangeable layer, modules were designed to make the layer dynamic, flexible and adaptable. The layer was both application and platform independent which allowd systems to reuse specific solutions that have already been developed.

It is essential to plan and design suitable cooperative mechanisms to provide adequate control over the behaviour of multiple agents systems. However, cooperative mechanisms proposed in various multi-agents systems are lack of common standard. It will be useful to define a clear system layer and cooperative agent functionalities in order to achieve the potential power of multiple agents systems. Effective cooperative mechanisms can be also a key element to keep the warehouse in synchronisation with distributed local systems.

3.2.5 Mobile Agent Based Self-Adaptive Join

One of the most related works using software agents to improve join operations is "Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing" by Arcangeli (J. Arcangeli, 2004). Arcangeli proposed a decentralised join execution model based on mobile agents, which were capable of deciding on their own location and move autonomously. The main idea of this work was based on mobile join execution model which utilised mobility of mobile agents to move the join operation around in order to improve traditional join algorithms.

Arcangeli's work demonstrated that software agents can be effectively utilised to address issues in wide-area distributed query processing. The research work has several aspects in common to our proposal to be introduced in the next chapter:

- Both aim to improve join performance
- Both are agent-based
- Both use adaptive mechanisms

The work also shows that as one of the most expensive operations in query processing, the issues of join operation and it performance improvement has been widely researched.

The main methods used in Arcangeli's work are:

- Execute join operations as close as possible to data
- Add mobile operators into traditional join algorithms

- Use statistical information, data availability, and system state to dynamic decide dynamically whether and where the join operation should be moved to in order to optimize join operation.

Arcangeli's work was very interesting and inspiring, but it had some noticeable weaknesses and limitations. Since the work was based on traditional join algorithms, join results would be only available when data from both join relations were received. It is not a preferable solution in the distributed environment. Dynamic estimates of statistical information, data availability, and system state as well as moving join operations could cause system overheads. The mobile operator was added into various join algorithms separately, therefore, it could not make full use of advantages of various join algorithms.

Arcangeli's work was suitable only when the 'errors' reached certain level. It was also reported in their work that the algorithm only worked effectively on the network speed below 1.28Mb/s (J. Arcangeli, 2004). It is quite low compared with the current network speed.

The work was evaluated under very restrictive environment. It assumed that the characteristics of all sites for join were the same, site failures and data unavailability were not considered, and agents buffer size was unlimited. In a wide-area distributed network, site failures and data unavailable are not uncommon phenomenon.

Since our aim is to provide efficient join algorithm for distributed data warehousing, it has to take into consideration of data warehouse features into the dynamic adaptive model. It

should work very efficiently both in a slower speed network environment and in high speed network environment.

3.3. A Proposed Framework for Agent-based Data Warehousing

In recent years, the data warehouse approach for distributed information management has been increasingly utilised with a varying degree of success. It facilitates the 'discovery' of information in organisations' OLTP systems and providing a better understanding of real-world problems. As such, it becomes a cornerstone to support decision-making and analytical data processing.

A data warehouse is a separate repository from data sources and information stored in a data warehouse is extracted and integrated from multiple data sources in advance. This data warehouse approach provides a potential for better query performance than other approaches. However, there exists a significant level of difficulty in keeping the data warehouse in synchronisation with various data sources which are often autonomous and dynamic. Although most analysis can be done monthly, weekly or daily, key decisions are increasingly required to be made based on current events to meet the business changes in real-time. For example, in the telecommunication, the communication routings are required to be updated based on current communication results to select a route for better quality with less cost. If the data used for decision making is too old, the results will become useless. Consequently, this requirement of continuous update may render data warehouses unpractical as data update and processing may take longer than required by real-time applications.

In addition, various information sources such as operational data in retailers or finance institutions are often autonomous and dynamic and generally used for daily transaction rather than supporting data warehouses. An important consequence of the autonomous data sources is the fact that those sources may change without being controlled from a higher data integration layer. Many sources, particularly Web-based data sources, may not only change their data, but also their capabilities without cooperating with users of their data.

Software agents can perform complex tasks on behalf of users and have unique properties of autonomy, adaptiveness, collaboration and mobility. They are suitable for use in wide variety of applications. In particular, they are well suited for applications which involve distributed computation or communication between components, sensing or monitoring of their environment, or autonomous operation. Based on our investigation and the feasibility study on software agent technology, we believe that the software agent technology can be utilised to address data synchronisation and query performance issues in distributed data warehouses.

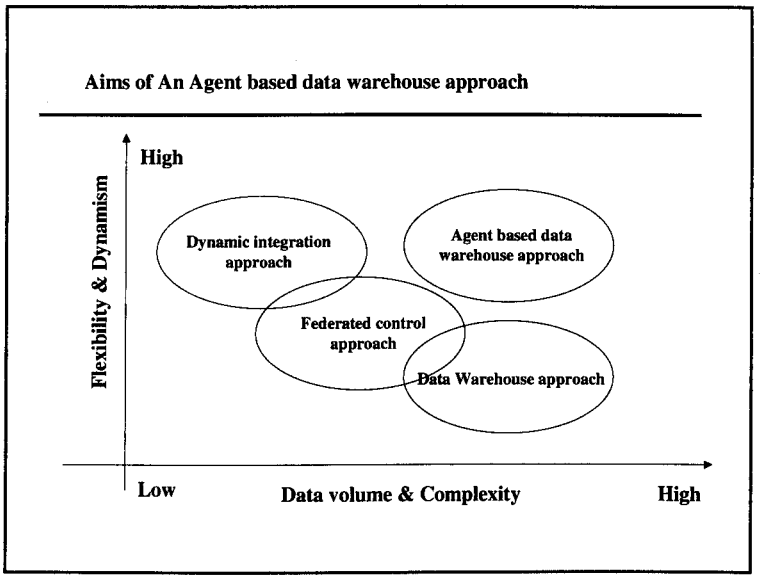

Figure 3.2 Objectives of the agent-based data warehouse approach

We propose an agent-based data warehousing (ABDW) approach, in which software agents are used to incorporate dynamic integration approach and traditional data warehousing approach seamlessly. The main aim of the ABDW approach is to manage large volume of complex data efficiently and to improve the flexibility and adaptability at the same time. The comparison among the other distributed information management approaches are depicted in Figure 3.2 above.

The core objective of the proposed ABDW approach is to address data warehousing issues arisen from distributed and dynamic data warehouse environments especially in dynamic integration, adaptive ETL, and distributed data access areas. In the ABDW approach, data warehousing functions are organised in a multi-agent platform where software agents are working intelligently and autonomously to adapt to dynamic environment and cooperating each other to achieve better performance.

3.3.1 Agent-based Data Warehousing Architecture

The architecture of the proposed ABDW is depicted in Figure 3.3, in which data warehousing functions are organised in a MAS platform. The ABDW architecture consists of the following components: agent-based ETL, agent-based data warehouse, agent-based user interface, and a MAS platform. They are working intelligently and autonomously to adapt to the distributed and dynamic environments. They are also working cooperatively together to provide better performance.

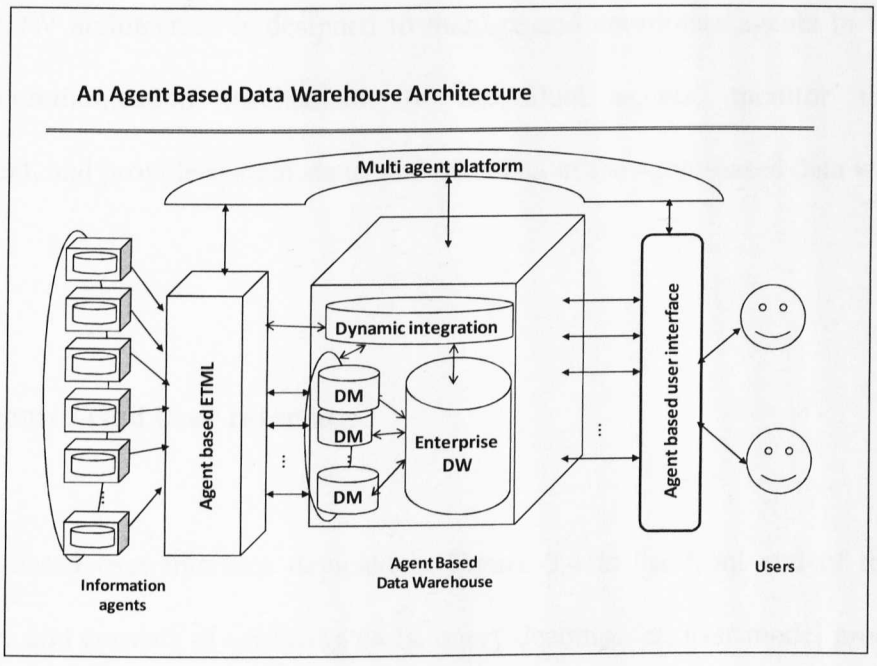

Figure 3.3 Agent-based data warehouse architecture

In the ABDW architecture, the agent-based user interface is facilitated with various data accessing tools for users to analyse data retrieved from agent-based data warehouse. It is also responsible for decomposition of complex requests received from users into smaller tasks for various agents. The agent-based data warehouse is the centre of the architecture and it consists of data marts, enterprise data warehouse, and dynamic integration component. Data marts are managed by information agents. Each data mart is focusing on

one subject or functional area and may reside on a remote server. Enterprise data warehouse stores pre-integrated data from data marts and some of atomic and detailed historical data are allowed to overlap with data in data marts to provide better query performance. The dynamic integration component is used for retrieve current/real time data and also responsible for integrating data. The agent-based ETL is designed to monitoring, extracting, transforming and integrating information from various data sources in to their local data marts in the agent-based warehouse. It is facilitated with information agents to realise adaptive ETL and dynamic integration. The MAS platform in the ABDW architecture is designed to manage and coordinate agents in the system, provide communication mechanism for individual agents, monitor the system environment, and provide system status and metadata of the agent-based data warehouse.

3.3.2 Agent-based User Interface

The agent-based user interface depicted in Figure 3.4 is the front end of the ABDW architecture and consists of accessing tools, query decomposer, user model processor and results presentation components. It receives requests form user through various user accessing tools and send relevant information into complex query decomposer and user model processor. In the query decomposer, users' complex requests are decomposed into several small tasks which can be carried out by individual agent available in the MAS platform. Those agents interact with the agent-based data warehouse to retrieve user required information effectively and efficiently. It will access data through dynamic integration component of the agent-based data warehouse if the current data is involved. Otherwise, data can be received directly from related data marts or the enterprise data

warehouse. Users' data accessing patterns will be processed and recorded into their own profiles by user model processor. The interface agents could complete various tasks on behalf of users autonomously according to the users' profiles to improve the efficiency. Once results received back from agent-based data warehouse, results presentation component will present results to users in their favourite format according to their profiles.

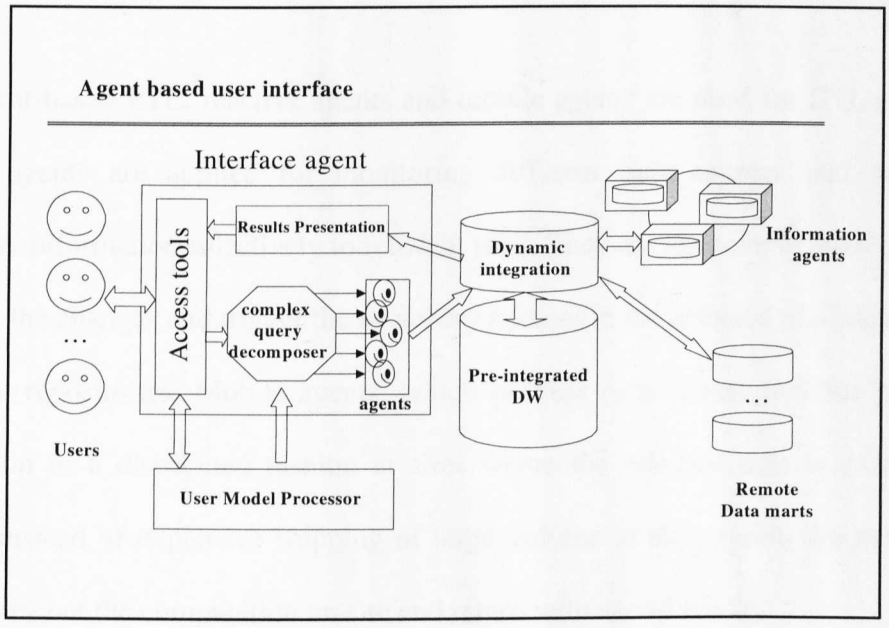

Figure 3.4 Agent-based user interface

The agent-based user interface offers users an intelligent and easy to use interface to improve usability and accessibility. The interface can hide from the actual underlying complex processing and can 'learn' from users' profile which is created based on users' behaviours to establish a user model which could be used to prepare results for users in advance to delivery better query performance.

3.3.3 Agent-based ETL

At the back end of the ABDW architecture, there is an agent-based ETL depicted in Figure 3.5, which is used to extract, transform, and load information from various data sources into the agent-based data warehouse. Since the data sources can be managed by information agents, mobile agents could be applied to improve efficiency of ETL processing.

In the agent-based ETL, reactive agents and mobile agents are used for ETL processing. Reactive agents are applied for monitoring different data sources and routing the appropriate information selectively to relevant sites. They are also employed to detect and propagate the changes and trigger the necessary updates in the affected pre-integrated data warehouse repositories. Mobile agents, which provide an efficient tool for performing computation in a distributed fashion at sites where the relevant data is extracted and prepared instead of expensive shipping of large volume of data across the network, are used to carry out the computation on site and return with useful results.

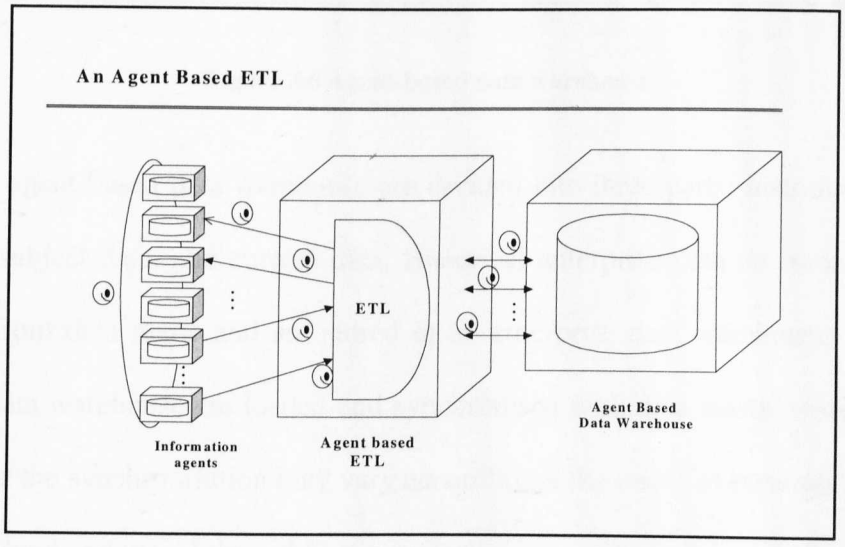

Figure 3.5 Agent-based ETL

3.3.4 Agent-based Data Warehouse

The agent-based data warehouse is in the centre of the ABDW architecture. It incorporates data marts, enterprise data warehouse and dynamic integrated component together to support both historical and current data effectively as shown in Figure 3.6.

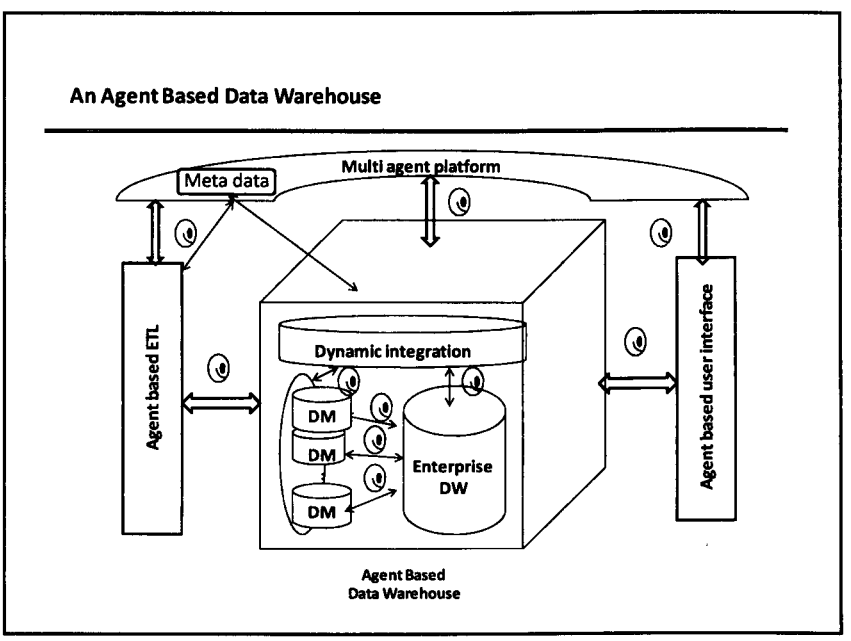

Figure 3.6 Agent-based data warehouse

Data in the agent-based data warehouse are divided into three parts: historical enterprise data, local subject data, and current data. Historical enterprise data are summarised and integrated from data marts and are stored in an enterprise data warehouse. Data in the enterprise data warehouse are loaded and synchronised with data marts periodically. The frequency of the synchronisation may vary according to the needs of business and volume of data involved and weekly/monthly is a typically option. Atomic and detailed historical

data are stored in data marts which are usually resided at local server directly connected with operational database server in order to reduce the cost of data transport during the ETL process. Since the performance of the operational database will be affected when queries on the database involves large amount of data, usually the ETL process will be run at less busy period such as midnight. Therefore, the current/runtime data will not be available at both the enterprise data warehouse and data marts. The dynamic integration component is designed to retrieve the current data allocated at remote data sources and to join them together with data retrieved from the enterprise data warehouse and data marts.

When users access the agent-based data warehouse, usually their queries will be decomposed into smaller tasks in the agent-based user interface mentioned in section 3.3.2. If a query involves with both current and historical data, it will be partitioned according to time. Since data warehouses or data marts always have a time dimension and the latest data available at data marts or enterprise data warehouse can be determined by the system frequency of data loading configuration, it is not an issue to determine partition of data. Traditional data warehouse queries can be used to retrieve the historical part of the data from the data marts or enterprise data warehouse. For the current part of the data, the query will be further processed in dynamic integration component, in which mobile agents will be used to retrieve data from remote information agents and join the historical part of the data to answer users' queries.

3.3.5. MAS Platform

The MAS platform is the core in the ABDW architecture. It is designed to manage and coordinate all agents in the system. It is responsible for initialisation of the system,

activate and deactivate individual agent, resource allocation among agents based on the users' requirements and system environment status. We believe in some degree that the ultimate goal of the agent technology is to make the full use of the available resources to achieve the best possible overall performance for a system. The MAS platform also used to provide an infrastructure for the specification of communication and interaction protocols which allow agents to exchange information and knowledge in order to work collaboratively together. The system environment is closely monitored and the related information such as size of data source, network speed, available memories and CPU capacity of individual server linked in the system etc. Meta data used for describing data source, data marts, enterprise data warehouse, and the transformation details are managed by the MAS platform. A service directory for agents to list their abilities and needs is also included in the MAS platform.

3.3.6. Key Techniques and Challenges in ABDW Architecture

The agent-based data warehousing approach aims to provide a seamless integration between per-integrated data warehouse and dynamic integration processing. It takes advantages of MAS platform, remote information agents and mobile agents to accomplish different tasks and join information efficiently. The ABDW approach could have the following main benefits: more effectively to deal with real-time data integration problem, more flexibility to adapt autonomous and dynamic data environment, and better data warehouse performance.

In order to make the ABDW architecture function effectively and efficiently as we expect, the following key techniques must be in place. Firstly, a MAS platform with suitable mechanisms should be designed to provide adequate coordination and control over individual agents which might act autonomously. Functions might include coordination and synchronisation among agents, activation and deactivation of individual agents or groups of agents, selection among agents, creation of new agents when needed, elimination of agents which are no longer needed, adaptation agents to changes in the environment, learning from experience. Although there is no ready to use MAS platform designed for ABDW, many software agent engineering methodologies and developing tools(M. Wooldridge, 2000), (N. R. Jennings, 2001), (Parandoosh, 2007) could be used to build up such platform.

Secondly, information agents may apply to as many data sources as possible so that data sources can be changed from passive objects for access into active information agents. They can provide intelligent, selective, and context-sensitive data gathering, transformation and integration functions for large scale data processing. Mobile agents could be used to work for both the agent-based ETL and the dynamic integration component so that most of tasks can be done at distributed remote sites locally. However, as the data sources could be autonomously owned and operated, it may not be always possible for information agents to reside on remote platform due to heterogeneous or security reason. The mobile agents should be able to adapt the change environment.

Thirdly, most of functions and principles of interface agents could be applied to the agent-based user interface in the ABDW architecture to improve the usability and efficiency of the interface. In addition, a complex query decomposer needs to be designed to

decompose queries into small tasks which agents can carry out. Coordination and communication mechanism among individual agent should be considered, which could be achieved by using Agent Communication Language (ACL) in MAS platform.

Dynamic integration will be the last but most important technique in the ABDW architecture. All necessary information will be joined together to produce final results in the dynamic integration component. Dynamic integration component receives requirements from interface agents and makes decision to retrieval information from data marts, enterprise data warehouse, and/or remote data sources based on whether required information is historical or current, summarised or detailed. Dynamic integration component may gather current data use mobile agents through the agent-based ETL, and use agents to obtain historical data directly from data marts and/or enterprise data warehouse according to the level of details required. All received data will be joined to produce the results for users.

To achieve the objectives set for the agent-based data warehousing approach, the main challenge is to provide a high performance of join data from remote data because the join performance is the key factor to address data synchronisation and query performance issues. Since data marts and data sources are often distributed at different sites connected via various network, the query performance from the remote sites is very difficult to predict due to dynamic data sources and variable network conditions. The join operation is one of the most expensive operations in query processing as it combines, compares and merges potentially large data sets. Consequently join performance has a considerable impact on overall system performance especially in a distributed warehouse environment.

It is essential to develop a novel join algorithm which could provide better performance in distributed and dynamic data environment.

Having proposed the ABDW architecture, it is not the objective of this research to implement the ABDW architecture in its entirety. Instead, the work is focused on the use of intelligent agents in population, maintenance, and query processing aspects of data warehouse, in particular, an agent-based join algorithm for such an architecture.

Chapter 4. JOIN ALGORITHMS IN DISTRIBUTED DATA ENVIRONMENT

Joins are the most frequently used operations among the basic relational operations (SELECT, JOIN, PROJECT). Relational database systems organise information into a collection of tables. A join operation must be used when any queries involves information from two or more tables. Joins are also the most expensive operations that a relational database system performs in terms of both time and memory. To join two large tables could consume a significant amount of the system's CPU cycles, disk or network bandwidth, and buffer memory (Bhashyam, 2004). Therefore, the performance of the join algorithms plays an important role in the overall query performance.

In a typical data warehouse dimensional data model, a large volume of organisation's data are organised into fact tables and dimensional tables. Fact tables have relationships with one or more dimensional tables. Data warehouse queries always involve join operation between fact tables and dimensional tables. One of the main objectives of a data warehouse is to provide better query performance for users to make a right decision at the right time. For this reason, effective and efficient join algorithms are a critical factor in determining the overall performance a data warehouse.

This chapter studies a range of typical and popular join algorithms and the factors that affect their performance in distributed data environments. The state of art of modern adaptive join algorithms is investigated. The main issues of the current adaptive join algorithms are highlighted. A novel agent-based join algorithm called AJoin for the agent-based data warehouses is proposed. It aims to utilise intelligent agents to coordinate a

ripple hash join and semi-join operations to adapt the change data environment to achieve the best query performance for data warehouses.

The rest of this chapter is structured as follows. Section 4.1 studies join algorithms for the relational database system. The strengths and weaknesses of each type of the join algorithm and the factors that affect performances of joins in distributed data environments are highlighted. Section 4.2 investigates the effectiveness of modern adaptive join algorithms in distributed and dynamic data environments and highlights the main issues of the current join algorithms in distributed data warehouse environments. Section 4.3 presents a proposed agent-based join algorithm.

4.1. Join Algorithms in Relational Databases

Due to the importance of join algorithms in a database system, a lot of research has been carried out and come up with many types of join algorithms (Coronel et al., 2009), (Rahimi & Haug, 2010), (Silberschatz et al., 2010).However, no single join algorithm is the best for all since the join performance is largely dependent on the input data and its environment. In this section, we study several main algorithms for computing the join of relations, and analyse their strengths and weaknesses and the factors that affect their performances in distributed data environments.

When joining relation R and S, every tuple in R needs to be compared with every tuple in S to see if the join condition across their attributes is satisfied. When the condition is met, the rows are concatenated and copied into the result relation. There are many approaches

for performing a join. We will analyse the nested-loop approach (with or without indexes), the sort–merge approach, and the hash-join approach.

In the thesis, we use the notation $R \bowtie_{R.a=S.b} S$ for the join operation, where a and b are join attributes or sets of attributes of relations R and S respectively.

4.1.1 Nested-Loop Join

The nested-loop join is a classical row-based approach to performing a join operation, which joins two relations R and S by making two nested loops. Based on (Silberschatz et al., 2010), the algorithm of the nested-loop join is listed in the following Java-style pseudo code in table 4.1.

```
For each r in R {                    // r is a tuple in R
        For each s in S {            // s is a tuple in S
                If (r.a = s.b)
                        output the tuple r||s
                // "||" represents a concatenation operation
        }       // next s
}               //next r
```

Table 4.1 Nested-loop join algorithm

The nested-loop join algorithm is expensive, since it examines every pair of tuples in the two relations. To perform the join using a nested-loop approach, we have to spend M disk I/Os to bring all the pages of R (one-by-one) into memory, where M denote the number of pages in R. For each tuple of R, and we have to spend N disk I/Os to bring all the pages of

S (one-by-one) into memory and then examine each tuple in S, where N denote the number of pages in S. Therefore, the number of disk I/Os required is O(M + nr* N), where nr denotes the number of tuples in R. In the best case, there is enough space for both relations to fit simultaneously in memory, so each block would have to be read only once; hence, only O(M+N) of disk I/Os would be required.

The nested-loop join algorithm can take advantage of additional memory to reduce the number of times that the S relation is scanned. If one of the relations can fit entirely in main memory, it can be used as the inner relation to reduce the join cost, since the inner relation would then be read only once. Therefore, if S is small enough to fit in main memory, the approach requires only a total M+N of disk I/Os, which is the same cost as that for the case where both relations fit in memory.

The most noticeable advantage of the nested-loop join algorithm is that the nested-loop joins can be used regardless of the complex of join conditions. It can be implemented with complex join conditions, such as conjunctions and disjunctions.

4.1.2 Block Nested-Loop Join

The simple nested loops join algorithm does not effectively utilise buffer pages. If the buffer is too small to hold either relation entirely in memory, we can still obtain a major saving in block accesses if we process the relations on a per-block basis, rather than on a per-tuple basis. Based on (Silberschatz et al., 2010), the algorithm of the block nested-loop join is listed in the following Java-style pseudo code in table 4.2.

The block nested-loop join a variant of the nested-loop join where every block of the inner relation is paired with every block of the outer relation. Within each pair of blocks,

every tuple in one block is paired with every tuple in the other block, to generate all pairs of tuples. As before, all pairs of tuples that satisfy the join condition are added to the result.

```
for each block Br of R {
        for each block Bs of S {
                for each tuple r in Br {
                        for each tuple s in Bs {
                                If r.a = s.b
                                  output the tuple r||s
                                  // "||" represents a concatenation operation */
                        }       // next s
                }               // next r
        }                       // next Bs
}                               // next Br
```

Table 4.2 Blocked nested-loop join algorithm

The cost of the in the block nested-loop join, worst case, each block in the inner relation s is read only once for each block in the outer relation, instead of once for each tuple in the outer relation. Thus, in the worst case, there will be a total of O(Br *Bs + Br), where Br and Bs denote the number of blocks containing records of r and s respectively. In the best case, where the blocks of outer relation can all be fitted in the memory, the cost will be reduced to O(Br + Bs). Therefore, the block nested-loop join algorithm is more efficient compared with the basic nested loop join algorithm,

4.1.3 Indexed Nested-Loop Join

In a nested-loop join, when an index is available on the attribute of a join relation, the performance of the finding the matching tuples can be improved.

Assuming there is an index on relation S, the cost of an indexed nested-loop join can be computed as follows. For each tuple in the outer relation R, a lookup is performed on the index for S, and the relevant tuples are retrieved. In the worst case, there is space in the buffer for only one page of r and one page of the index. Then, M disk accesses are needed to read relation R, where M denotes the number of pages in R. For each tuple in R, we perform an index lookup on R. Then, the cost of the join can be computed as O(M+nr *C), where nr is the number of records in relation R and C is the cost of a single selection on S using the join condition.

Since only one of the indexes can be used, there will be no different in terms of the join cost.

4.1.4 Sort-merge Join

In this join approach, the two relations involved are sorted based on the join attribute and then the sorted relations are merged. The overall cost of the join is the sum of the sort cost and the merge cost. Based on (Coronel et al., 2009), the algorithm of the sort-merge join is listed in the above table 4.3.

```
if R not sorted on attribute a, sort it;
if S not sorted on attribute b, sort it;
Tr = first tuple in R;
Ts = first tuple in S;
Cs = Ts          // start of current S-partition
while (Tr != eof and Cs != eof) {
        while (Tr != eof and Tr.a<Cs.b)
                Tr = next tuple in R;
        while Cs != eof and Tr.a>Cs.b
                Cs = next tuple in S;
        while Tr != eof and Tr.a==Cs.b {
                Ts=Cs;
                while Ts != eof and Tr.a==Ts.b {
                        output the tuple r||s ;
                        // "||" represents a concatenation operation
                        Ts=next tuple in S;
                }
                Tr= next tuple in R;
        }
        Cs=Ts;
}
```

Table 4.3 Sort-merge join algorithm

In practice, the most expensive part of performing a sort-merge join is arranging for both inputs to the algorithm to be presented in sorted order. This can be achieved via an explicit sort operation or by taking advantage of a pre-existing ordering in one or both of the join relations. The latter condition can occur because an input to the join might be

produced by an index scan of a tree-based index, another merge join, or some other plan operator that happens to produce output sorted on an appropriate key.

Let's say that we have two relations R and S and | R | < | S |. R fits in M pages memory and S fits in N pages memory. So, in the worst case Sort-Merge Join will run in O(M + N) I/Os. In the case that R and S are not ordered the worst case will be O(M + N + 2(M + Mlog(M) + N + Nlog(N))), where M and N denote the number of pages in R and S respectively.

4.1.5 Hash Join

In this join approach shown in Figure 4.1 based on (Ramakrishnan & Gehrke, 2002); a hash function *h* is used to partition tuples of both relations. The hash-join approach consists of two phases: the partitioning phase and the probing phase. In the partitioning phase, the tuples in each relation R and S is partitioned with the same hash value on the join attributes into the two separate sets of non-overlapping partitions. A tuple is assigned to a particular partition by using the same hash function *h* for both relations. When partitioning R, attribute "a" is passed to the hash function, while the partitioning of S passes attribute "b." Since we use the same hash function for both relations in the partitioning phase, the matching tuples from both relations, if any, end up in the buckets with the same address. In the probing phase of the join, tuples in a partition of R are only compared to the tuples in the corresponding partition of S.

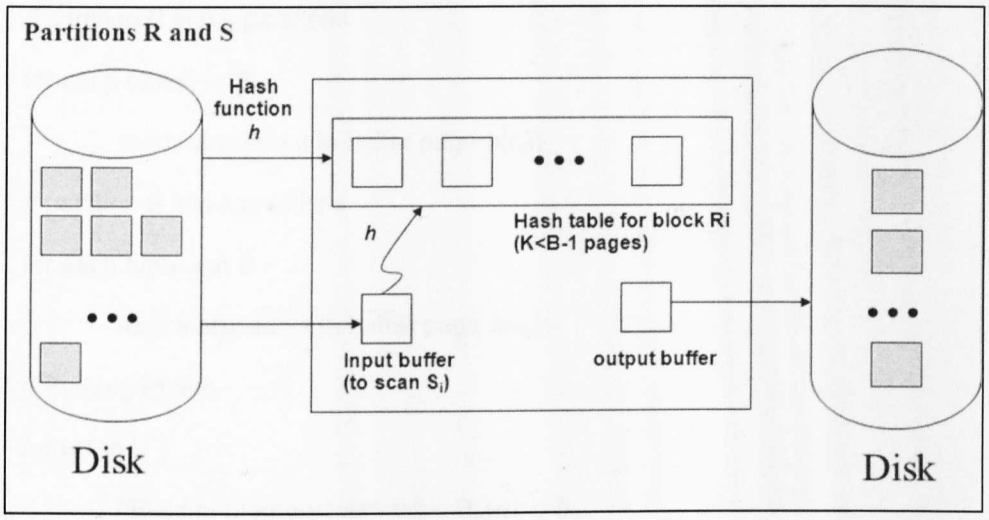

Figure 4.1 Hash join approach (Ramakrishnan & Gehrke, 2002)

Assume that hash function h can partition R and S into k partitions and k<B-1 pages and B<$\sqrt{f \times M}$ where B, M and f denote the buffer pages, the number of pages in R, and a fudge factor respectively. Based on (Ramakrishnan & Gehrke, 2002), the algorithm of the hash join is listed in the following Java-style pseudo code in table 4.4.

The cost of hash join is the total cost of partitioning both relations and the cost of probing. In the partitioning phase, each relation is scanned and is written back to the disk. Therefore, the partitioning phase cost is O(2 * (M + N)), where M and N denote the number of pages in R and S respectively. In the probing phase, each relation is scanned once again. Hence, the total cost of a hash-join is O(3 * (M + N)). Compared to the nested-loop join approach, the performance of the hash join is much better. It has been increasingly popular in today's DBMSs. However, the hash join algorithm can be used to implement natural joins and equi-joins, but it cannot support other complex joins.

```
// partition R into k partitions
for each tuple r in R
        read r and add it to buffer page h(r_a);
// partition S into k partitions
for each tuple s in S
        read s and add it to buffer page h(s_b);
// Probing Phase
for i=1 to k  {
        //Build in-memory hash table R_i,using h
        for each tuple r partition R_i,
                read r and insert into hash table using h(r_a);
        //Scan S_i,and probe for matching R_i, tuples
        for each tuple s partition S_i, {
                read s and probe table using h(s_b);
                for each matching tuple r in R_i,
                        output the tuple r||s ;
                        // "||" represents a concatenation operation
        }
        clear hash table to prepare for next partition;
}
```

Table 4.4 Hash join algorithm

The performance of the hash join can be affected considerably, when hash table overflow occurs. Hash table overflow can occur if there are many tuples in the build relation with the same hash values for the join attributes, or if the hash function does not have the properties of randomness and uniformity. In either case, some of the partitions will have more tuples than the average, whereas others will have fewer. This phenomenon is call partition skew. When the partition skew is exceeding the level where one of the partitions

for a join relation is larger than available memory for the join operation, hash table overflow occurs.

If the level of partition skew is low, the problem can be handled by increasing the number of partitions so that the expected size of each partition will not beyond the size of available memory. The number of partitions will be increased by a small value called the fudge factor mentioned previously, which is usually about 20 percent of the number of hash partitions computed according to (Silberschatz et al., 2010).

Hash table overflow may still occur if the level of partition skew is high. To address to problem, either overflow resolution or overflow avoidance technique could be used. Overflow resolution is performed during the build phase, if a hash table overflow is detected. Overflow resolution proceeds in this way: a different hash function will be used to further divide the large partition into smaller partitions. The new hash function will be applied to both relations on the same partition. In contrast, overflow avoidance performs the partitioning carefully, so that overflows never occur during the build phase. In overflow avoidance, the build relation S is initially partitioned into many small partitions, and then some partitions are combined in such a way that each combined partition fits in memory. The probe relation R is partitioned in the same way as the combined partitions on S.

4.2. Join Algorithms in Distributed Database Systems

A distributed database system allows applications to access data from local and remote databases. In a distributed database system, data can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. To join data in a distributed database system is usually much more expensive in term of time and resource.

Compared the cost of communication with the local data processing in disk and memory, the communication cost is usually much higher. A current widely used standard for the Serial Advanced Technology Attachment (SATA) disk "buffer-to-computer" interface is 3.0 Gbit/s, which can send about 300 megabyte/s with 10 bit encoding. A typical 100M network can transmit maximum 12.5 megabyte/s theoretically. According to a recent (November 2011) survey from a technical support staff from Virgin Media – the one of the UK's fast business broadband provider, the current maximum speed limit for VPN via Internet other than leased line is 5 Mb/s. Therefore, the communication cost is one of the most important factors to consider in the cost evaluation of joins in a distributed database system.

Join algorithms investigated in the last section 4.1 are focused on conventional centralised database systems, where no communication cost consideration is required. The cost of join is calculated based on disk I/O and the join performance optimisation is achieved by reducing the disk I/O requirements. It is not suitable for join algorithms in distributed database systems where communication cost is the main factor of overall join cost.

4.2.1 Semi-Join

To minimises the communication cost between sites in a distributed environment during the join operation, semi-join was invented and used in (Bernstein & Chiu, 1981)'s project and then extended and applied by (Apers et al., 1983). A semi-join involves an increase in local processing of data compared with a conventional join, but saves on the cost of data transmission between sites.

Semi-join attempts to qualify the tuples before the relations are actually joined. In distributed systems, a semi-join qualifies the tuples in both relations that match the join conditions before sending the tuples across the network. As a result, tuples that are not part of the final join results will not be transmitted, which reduces the communication costs.

According to (Bell et al., 1992), the semi-join can be formally defined as follows:

$$R \ltimes S = \pi_i (R \bowtie S) \qquad (4.1)$$

where R and S are two relations stored at remote sites S1 and S2 respectively, the symbol $\ltimes$ as the denotation of the semi join operation, and π_i denotes the projection over the attributes R_i of R.

The above formula can be replaced by the equivalent form:

$$R \ltimes S = R \bowtie \pi S \qquad (4.2)$$

where π is over the join attributes only.

The formula represented in (4.2) has potential advantages over the (4.1). According to (Bell et al., 1992), a full join using semi-join operation has two phases: reduction phase and processing phase. In reduction phase, the semi-join projects the join attributes from S at site S2 (=π S) and transmits π S to site S1. In the following processing phase, the semi-

join computes the join results R ⋈ π S and sends them the join results site. The algorithm of the semi-join is listed in the following table 4.5.

1. Project the join attributes R_a from R at S1to a temporary relation RT
2. Transmit RT from site S1 to site S2
3. Compute the join $RT \bowtie_{R.a=S.b} S$ at site S2 producing temporary relation ST
4. Transmit ST from site S2 to site S1
5. Compute the join $R \bowtie_{R.a=S.b} St$ producing the final results
6. Transmit results to results output site

Table 4.5 Semi-join algorithm

In step 1, a one-column relation RT will be produced form projection of join attributes Ra. The reason for doing this is that join attribute Ra is the only thing that needed to qualify the tuples in S. In step 2, the data transmission reduction occurs. Compared with a conventional join, the semi-join only transmits the join attribute Ra of R to site s2 where S is instead of sending the entire relation R. The communication cost of transmitting the join attribute Ra is much cheaper than transmitting a whole relation across the network. In step 3, RT and S are joined at site S2 and produced a temporary relation ST. At this stage, all of the tuples that qualify the join condition are identified. However, at moment, those non-join attributes of relation R is not presented at the temporary relation ST. To include any attributes required for output for these attributes, in step 4, ST will be transmitted back to site S1where R is. ST only includes those joined tuples is much smaller than relation S as any tuple that is not part of the final answer will not be transmitted. In step 5, S1 with R are joined to produce the final results. Final step 6, the full join results will be transmitted to results output site.

4.2.1 Semi-Join Cost vs. Benefits

The advantages of the semi-join algorithm are reaching greatest potential when the join attributes is relatively small and local processing costs are much smaller than transmission costs. Joins can increase the size of the result relation from the either join relation, whilst semi-join never does because addition attributes will not be added to the results.

The data need to be transmitted in a conventional join is the minimum size of the join relations (e.g. min(|R|, |S|)). While data need to be transmitted in a semi-join will be the minimum size of join attributes plus the size of semi-join (e.g. min(($|R_a|$+| R $\ltimes$S|), ($|R_s|$+| S $\ltimes$R|))).

Therefore, a semi-join will have a lower transmission cost than a conventional join if the cost of initiating a message, which is the extra cost to be included for semi-joins and should not be neglect based on (Kang & N. Roussopoulos, 1987) report, added to min(($|R_a|$+| R $\ltimes$S|), ($|R_s|$+| S $\ltimes$R|)) is less than min(|R|, |S|).

4.3. Adaptive Join Algorithms for Dynamic Data Environment

With the advances in Internet and web technologies, traditional dedicated leased line based private networks are replaced by VPN which uses the Internet to provide a cost effective solution to connect distributed networks to support organisations global operations. Data management for these organisations become more challenging due to unpredictable and dynamic natural of the Internet.

Join operations are very effectively optimised in conventional database management systems, which leveraging I/O cost information as well as histograms and other statistics to determine the best executable plans. However, databases systems distributed over the Internet presented a strong demand for new adaptive join techniques. The query optimiser in conventional database management systems may be no longer able to obtain necessary information for the query plan due to the unpredictable and variable network connections over the internet.

To address the challenges that arise from wide-area distributed network environments such as the Internet where data access becomes less predictable due to link congestion, load imbalance, and temporary outages, some modern pipelined join algorithms have been proposed by (P. J. Haas & Hellerstein, 1999), (Avnur & Hellerstein, 2000), (Urhan & M. J. Franklin, 2000), (Luo et al., 2002), (Deshpande et al., 2004). The basic principle of such modern adaptive join algorithms relies on the join being pipelined whereby operators in the query plan are executed in parallel. This could provide better response time as intermediate join results can be produced as soon as tuples from join relations are received and matched. These algorithms focus on using runtime feedback to modify query processing in a way that provides better response time or more efficient CPU utilisation.

In order to better understand and investigate the effectiveness of those modern adaptive join algorithms, the following typical modern adaptive join algorithms, such as nested-loop ripple join, Hash Ripple Join, XJoin has been investigated in depth.

4.3.1 Nested-loops Ripple Join

In conventional join algorithms, the goal is to minimise the completion time. The performance of conventional join algorithm will significantly affected when tuples received from either relation becomes very slow and uncertain. It means users might need to wait very long time to receive a complete relation before the first tuple can start join operation.

To tackle the problem, (P. J. Haas & Hellerstein, 1999) proposed nested-loop ripple join. The join can start as soon as one of tuples each received from both relations. The idea of ripple join is starting join operation once tuple was received instead of starting join operation until one relation completed. Figure 4.2 shows the "square" version of nested-loop ripple join, where R and S represent two join relations and n is the tuple received to be joined.

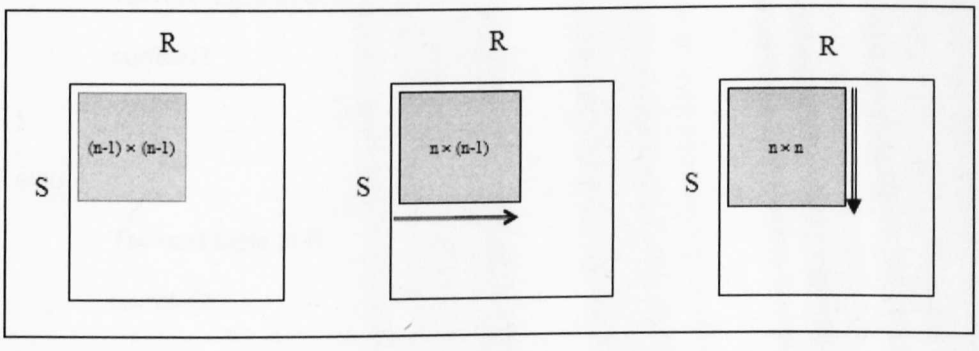

Figure 4.2 The "Square" version of nested-loop ripple join

The nested-loop ripple join alternates receiving from each of its input relations. When it receives a new tuple from one relation, that tuple is combined with all previously seen

tuples from the other relation. The square version of nested-loop ripple join algorithm can be described in Java-style pseudo code in table 4.7.

```
Tr=first tuple in R;
Ts=first tuple in S;
currel=S;
cur=1 ;
while (Tr !=eof and Ts != eof) {
  for ( i=1 to cur-1 )
          if Tr(i)a==Ts(cur)b
           output the tuple Tr(i)||Ts(cur);  // "||" represents a concatenation operation
  for (i=1 to cur)
          if Tr(cur)a==Ts(i)b
           output the tuple Tr(cur)||Ts(i);  // "||" represents a concatenation operation
  if (currel==S) {                                  // get net tuple, swap relation
          Ts=next tuple in S;
          currel=R;
  }
  else {
          Tr= next tuple in R;
          currel=S;
  }
}
```

Table 4.6 Nested-loops ripple join algorithm

As relation R and S may not receive at the same frequent, the sampling step of nested-loop ripple join may change into "rectangular". The ratio at which the join receives from

the two relations is critical to performance, and can be determined and modified dynamically by observing the statistical properties of the sets of tuples received so far.

In the worst case the performance of the nested-loop ripple join would be equivalent to a conventional nested-loop join. In fact, the nested-loop ripple join is a generalisation of nested loop join in which the traditional roles of inner and outer relation are continually interchanged during processing. Similar to the conventional nested-loop join, the performance of the nested-loop ripple join could be further improved by using block, index and hash techniques.

4.3.3 Hash Ripple Join

Nested loop join requires each tuple from one relation to match all available tuples from another relation. For equi-join query, it is natural to consider using hash technique to improve its performance. The idea is to hash both relations on the join attributes using the same hash function. Hash Ripple Join proposed by (P. J. Haas & Hellerstein, 1999), (Ives et al., 1999), (Luo et al., 2002) also called symmetric hash join, uses two hash tables from both relations instead of one and probes join relations with each other as well.

The basic idea of Hash Ripple Join is illustrated in Figure 4.3. When a new tuple (e.g. Tr) is received from one relation (e.g. R), it probes old tuples held in hash table Hs from another relation S to find the matches and then insert into hash table Hr. Next tuple (Ts) may receive from another relation S. then similarly it probes old tuples hold in hash table Hr from another relation R to find the matches and then insert into hash table Hs. Processing continue until all tuples are received and processed.

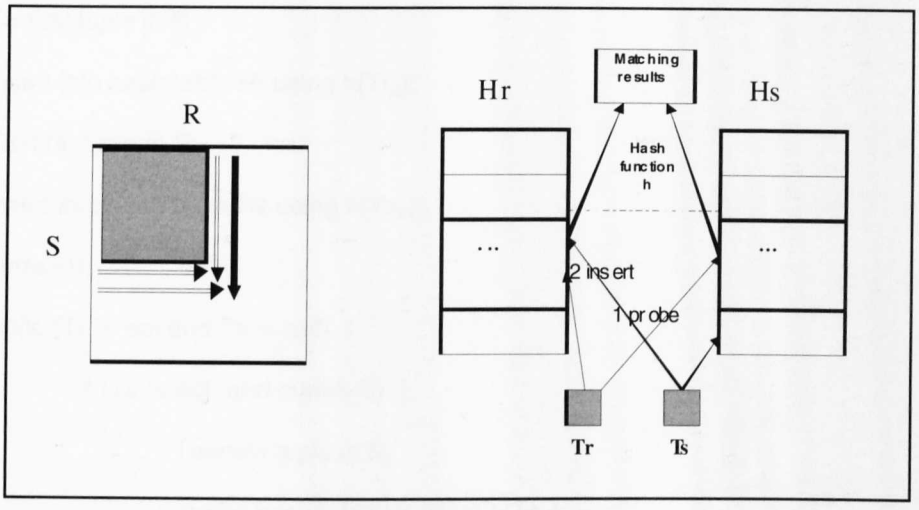

Figure 4.3 Hash Ripple Join

Similar to nested-loop ripple join, relation the sampling step of Hash Ripple Join may change into "rectangular" when R and S are not receive at the same frequent. The ratio at which the join receives from the two relations may be changed dynamically at runtime to achieve better join performance. The square version of Hash Ripple Join algorithm can be described in Java-style pseudo code in table 4.7.

Hash Ripple Join could provide an excellent join performance, but it requires sufficient memory space to hold both hash tables. Otherwise, memory may overflow and cause the join process to crash (Luo et al., 2002).

```
Tr=first tuple in R ;
insert into hash table Hr using h(Tr_a);
Ts=first tuple in S;
insert into hash table Hs using h(Ts_b);
currel=S;
while (Tr != eof and Ts != eof) {
        if (Ts != eof  and currel=S) {
                Ts=next tuple in S;
                 probe hash table Hr using h(Ts_b);
                for matching R tuples Tr(i),        // Tr(i)_a==Ts_b
                    output the tupleTr(i)||Ts;
                        // "||" represents a concatenation operation
                insert into hash table Hs using h(Ts_b);
        }
        if (Tr != eof  and currel=R) {
                Tr=next tuple in R;
                 probe hash table Hs using h(Tr_a);
                for matching S tuples Ts(i)// Tr_a==Ts(i)_b
                output the tuple Tr||Ts(i);  // "||" represents a concatenation operation
                 insert into hash table Hr using h(Tr_a);
        }
         if (currel=S )     // swap relation
                currel=R
         else
                curtel=S
}
```

Table 4.7 Hash Ripple Join algorithm

4.3.4 XJoin

As Hash Ripple Join requires that the hash tables for both of its inputs are kept in main memory during most of the query execution. As a result, the Hash Ripple Join may not suitable to use for joins with large inputs, and the ability to run multiple joins is severely limited. To overcome this problem, XJoin proposed by (Urhan & M. J. Franklin, 2000) extends the Hash Ripple Join to use less memory by allowing parts of the hash tables to be moved to secondary storage. XJoin does this by partitioning its inputs, similar to the way that hybrid hash join solves the memory problems in classic hash join.

The main difference between XJoin and Hash Ripple Join is that in order to reduce the memory usage, XJoin divided the hash partition into memory and disk two parts depicted in Figure 4.4. Newly received tuples will be hashed and stored in the memory part of the partition. When memory part becomes full, tuples in memory part of the partition will be flushed to disk. In order to join tuples in the both parts, XJoin divided join process into three stages.

At the first stage, it works similarly to the Hash Ripple Join, which joins only the memory part of partitions so that first tuple could be start join process as soon as a tuple received. The join processing will continue until all tuples have been received. Once all tuples have been received from both relations, XJoin starts third stage, in which XJoin joins memory parts and disk part as well as disk part and disk part to produce complete results. The XJoin stage 1 algorithm can be described in the following Java-style pseudo code in table 4.8.

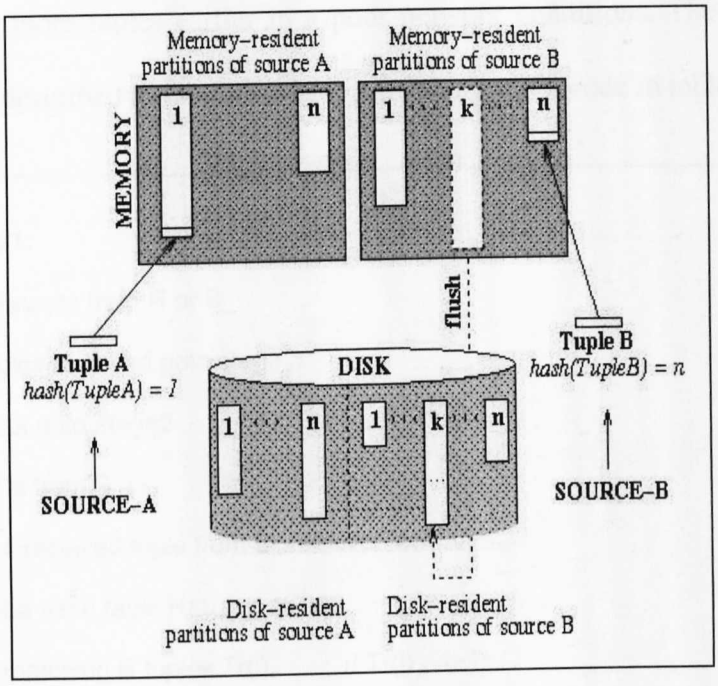

Figure 4.4 Partition handling from (Urhan & M. J. Franklin, 2000)

However, during the first join stage, join processing might be pause because of unexpected network delay which causes no tuples could be received from both relations. In such case, in order to improve XJoin performance, the second stage will be activated to produce join results. In this stage, the tuples from one relation in the memory part of the partition will be to probe tuples from another relation in the disk part of the partition. Once it has done, XJoin will go back to check if any tuple has received. If it has, then XJoin will return to first stage and continue the processing. Otherwise, it will carry on the second stage to probe next available partition. As the second stage works only when the first stage is not in progress, the work can be viewed as free. This is where the benefit of the second stage comes in. The risk is that when one or both of the inputs become unblocked it is not noticed until after the current disk-resident partition has been fully processed. In this case, the overhead of the second stage is no longer completely hidden. The reactively scheduled background process of XJoin provided an opportunity of utilises

delays to produce more tuples earlier in a poor network conditions. The XJoin stage 2 algorithms can be described in the following Java-style pseudo code in table 4.9.

```
XJoin_stage1:
    waiting inputs from R or S
    if all tuples received goto stage 3
    if timeout goto stage2
    if input is from S {
        Ts = received tuple from S
        probe hash table Hr using h(Ts_b);
        for matching R tuples Tr(i),        // Tr(i)_a ==Ts_b
                output the tupleTr(i)||Ts;  // "||" represents a concatenation operation
        if (h(Ts_b) in Hs is full )
                flush h(Ts_b) in Hr to disk Ds;
        insert into hash table Hs using h(Ts_b);
    }
    else {
        Tr = received tuple from R
        probe hash table Hs using h(Tr_a);
        for matching S tuples Ts(i)    // Tr_a ==Ts(i)_b
                output the tuple Tr||Ts(i);  // "||" represents a concatenation operation
        if (h(Tr_a) in Hr is full )
                flush h(Tr_a) in Hr to disk Dr;
                insert into hash table Hr using h(Tr_a);
        }
loop stage1
```

Table 4.8 XJoin algorithm – Stage 1

```
XJoin_stage2:
    for each tuple Tr in Dr or Ts in Ds {
        if (Ts in Ds ) {
            probe hash table Hr using h(Ts_b);
            for matching R tuples Tr(i) {        // Tr(i)_a ==Ts_b
              detect duplicate;
              output the tupleTr(i)||Ts;  // "||" represents a concatenation operation
            }
        }
        else {
            probe hash table Hs using h(Tr_a);
            for matching S tuples Ts(i) {   // Tr_a ==Ts(i)_b
              detect duplicate;
              output the tuple Tr||Ts(i);  // "||" represents a concatenation operation
            }
        }
        if inputs are ready goto XJoin_stage1;
    }
```

Table 4.9 XJoin algorithm – Stage 2

The multiple stages of XJoin may produce spurious duplicate tuples because they can perform overlapping work. Duplicates can be created in both the second and third stages. However, this problem has been addressed in XJoin by using a duplicate prevention mechanism based on timestamps. The XJoin stage3 algorithm can be described in the following Java-style pseudo code in table4.10.

```
XJoin_stage3:
For each tuple Tr in partition Ri  {
        If memory is not enough for a complete partition Ri
                flush other partitions to disk;
        read all Tr in partition Ri from Dr
        probe hash table Hs and Ds using h(Tra);
    for matching S tuples Ts(i)  {   // Tra ==Ts(i)b
          detect duplicate;
          output the tuple Tr||Ts(i);  // "||" represents a concatenation operation
        }
}
```

Table 4.10 XJoin algorithm – Stage 3

Compared with the conventional join algorithm, modern ripple join algorithms demonstrate better adaptivity to the changing environment. However, due to distributed and dynamic nature of modern ripple join algorithms, the traditional method of analysing join algorithms based upon the time required to access, transfer and perform the relevant CPU-based operations on a disk page (Harris & Ramamohanarao, 1996) no longer be able to provide an effective evaluation of join algorithms.

In order to study modern adaptive join algorithm performance in depth, the empirical research method is used to evaluate the effectiveness and performance of those modern ripple join algorithms based on evidence gathered from information collected from

experiments and observations. A detailed evaluation report based on experimental results of modern adaptive join algorithms is presented in the next chapter.

Chapter 5. Experimental Study on Modern Adaptive Join Algorithms

Empirical research works by the process of induction. Induction is the formulation of general theories from specific experiments and observations (Goddard & Melville, 2004). In order to make experiments systematic and purposeful, the experimental results must be comparable and based on typical scenarios. In this research work, a controlled experiment environment based on a data warehouse scenario and simulated network are established and used.

This chapter describes the data warehouse scenario used as a basis to investigate and evaluate various modern join algorithms for data warehouses. In particular, a distributed data warehouse architecture and multidimensional data model for the data warehouse scenario is discussed. This chapter also presents simulated network environments for the experiments of modern adaptive join algorithms. A comparative study on modern adaptive join algorithms is conducted. The need for an effective and efficient join algorithm for current distributed and dynamic data warehouse environments is highlighted.

The rest of this chapter is structured as follows. Section 5.1 describes the data warehouse scenario used as a basis to investigate and evaluate various modern join algorithms for data warehouses. Since data warehouses are typically used by large organisations, such as banks, retailer chains, insurance and telecommunication companies, our scenario is based on a typical telecommunication data warehouse system. The distributed data warehouse architecture and multidimensional data model of the data warehouse scenario are also discussed. Section 5.2 presents simulated network environments for the experiments of

modern adaptive join algorithms. Four types of network behaviours, such as low speed, random, high speed, and bursty are modelled. Section 5.3 discusses the experimental results and evaluates the effectiveness and performance of modern adaptive join algorithms.

5.1. A Data Warehouse Scenario for the Investigation

In order to investigate and evaluate effectiveness of data warehouse approaches and modern join algorithms in a data warehouse environment, a typical data warehouse environment based on real industrial case has been established.

Data warehouses have gained an increasing popularity among organisations which seek to utilise information technology to gain a competitive edge in today's global economy. It is particular helpful for organisations which have the sheer volume and complexity of data available from their various OLTP systems. Banks, retailer chains, insurances and telecommunication companies are typical users. Different users have different types of information available and concerns, but the principle and methodology of data warehousing for all type of systems are basically the same. For this project, a case scenario based on a telecommunication company is chosen as a typical example of data warehouses to discuss issues arisen from distributed and dynamic environments.

I.T.S is an international telecommunication company operated mainly in Europe. It has several sub-companies located in UK, Germany, France, Italy, Spain and Portugal. They are daily deliverying millions of calls for customs to all over the world. Without

compromising generality, the operations of the company are representative of the telecommunication industry including large volume of data, and a distributed and dynamic environment. As current telecommunication markets are very competitive and dynamic, it is vital to provide the management with an effective decision-making support system which can deliver the right information, to right people, at the right time, interactively and securely. Furthermore, data available from their OLTP systems are allowed to be used for the investigation.

The main OLTP systems running in the telecommunication system are switch systems which connect customer calls to various telecommunication carries and record all call details such as CLI (caller's line identification), destination zone, destination phone number, carrier's name, call duration, call charge, and so on. As a switch system is required to work very efficiently in order to make high volume of transactions successful, the OLTP database for the system only store information for the current day and old transaction details are archived in daily log files. In order to access the information, the daily log files are required to be imported into a separate database where OLAP runs. However, as data stored in log files is denormalised, to store all historical data directly into a database is not suitable for supporting daily business such as managing customers' account and providing billings. Usually, only one-year worth data will be kept in the database system and rest of data needs to backup to other storages. Another problem is that information is separated into various log files which may not fit the needs of decision making. To join information directly from distributed sources could lead to poor performance. In order to deal with the problems, a data warehouse system has been introduced.

5.1.1 Data Warehouse Architecture for the Scenario

The main business operation of the company is geographically distributed and a topology of I.T.S DW system is showing in Figure 5.1. Global DW and local DWs are connected in a WAN environment via VPN. As all sites run the same type of OLTP system to support daily business operations, the local DWs are basically using the same data schema too. The OLTP system is often required to update its function to meet the business requirements. However, usually those updates are not able to take place concurrently and have to implement from one site to another for the operation reason. Those updates may cause the change in data format of the log files. There are also some semantic differences in various sites. For an instance, the currency of call charge in UK site uses Pounds, but the currency of call charge in France site uses Euro. We can assume that all sites are running autonomously although in other data warehouse cases the situation may be much complex.

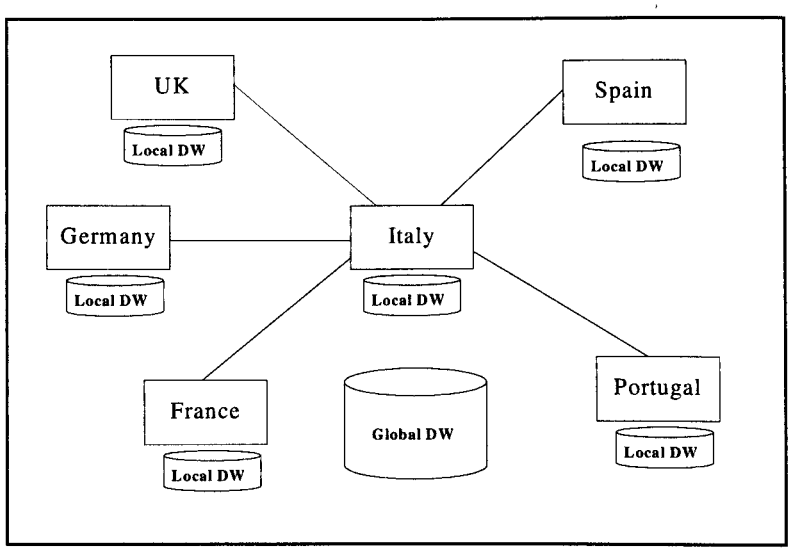

Figure 5.1 A topology of I.T.S DDW

A distributed data warehouse (DDW) architecture discussed in chapter 2 is used for the company. The architecture of the distributed data warehouse is illustrated in Figure 5.2.

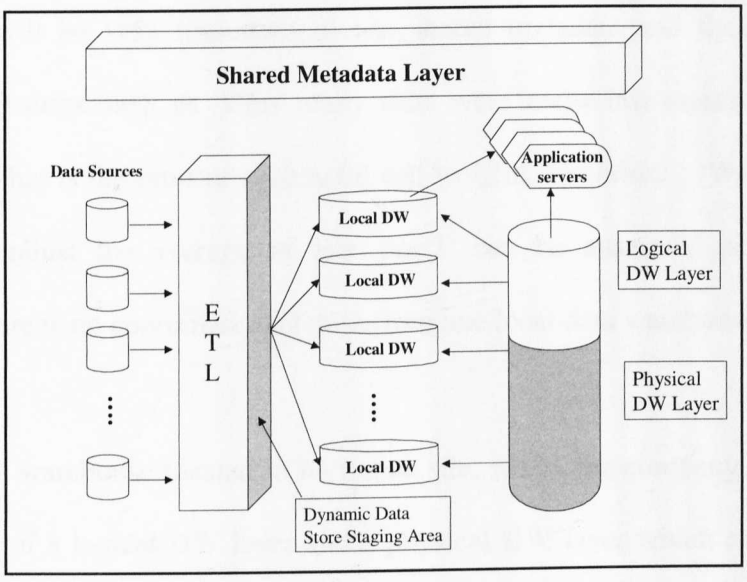

Figure 5.2 A DDW architecture

The architecture consists of local data warehouses (could also be viewed as data marts) and a global data warehouse. The local data warehouse represents data and processing at a remote site, and the global data warehouse represents that part of the business which is integrated across the business. Most amount of processing occurs at the local level. For instances, product sales, payment management, and routing selection are all processed and managed locally. As far as transaction processing is concerned, the local sites are running autonomously. Only for certain types of processing, such as a corporate balance, carrier payments are required at global level.

The local data warehouse located at each site contains data that is of interest only to the local level. Each local data warehouse has its own technology, its own data and its own processor where data from local log files will be extracted, transformed and loaded. As

the business operations in various sites are basically the same, similar data model could be apply to various sites and metadata could be shared easily. The currency difference may be the only difference between UK site and other Europe sites. However, in general data warehouse cases, the local data warehouse at various sites might be very different and autonomy. It will be very important to use shared metadata and dynamic data store staging area. Queries such as 'How many calls were made last month by a particular customer?', 'What is the ratio of successful call to USA last hour?', 'What is last month performance against the average of last year?' can be answered at the local data warehouse. There is no coordination of data from one local data warehouse to another.

The global data warehouse located in its Italian site, where the company's headquarter is based, consists of a logical DW layer and a physical DW layer which contains data that needs to be managed globally. It aims of the global DW to answer queries such as 'What is the total profit made by the whole corporation?' and 'How much should be paid to a particular telecomm carrier?' Queries could be answered transparently by the integration of physical DW and the logical DW which maps a unified logical data model to physical tables in various local DW together with caching and integrated services. The data is integrated across the corporation at the corporate level, such as summarised billing information, carrier information, and so on. The source data can be integrated from local data warehouse, directly from local log files, or obtain from headquarter database system.

5.1.2 Multidimensional Data Model

In order to organise large amounts of consolidated data for effective and efficient data analysis and retrieval, a multidimensional data model is used for the company's data

warehouse. An example of the multidimensional data model for the data warehouse is illustrated in Figure 5.3.

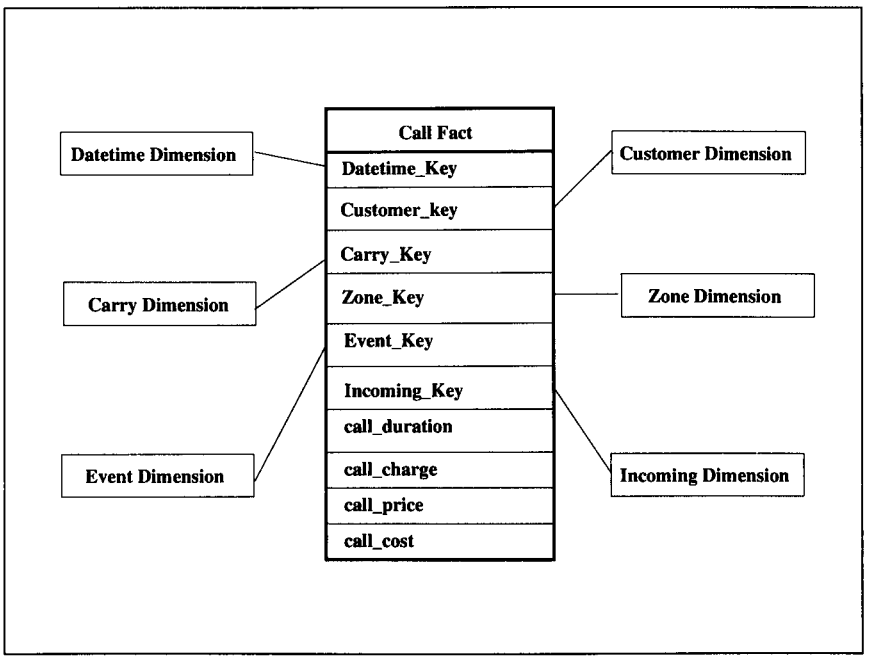

Figure 5.3 Multidimensional data model for I.T.S DW

In the multidimensional data model, the data in the data warehouse is stored in a denormalised manner to reduce the needs of join at run-time in order to improve query performance. In the data model the fact table stores the measurements of each call, such as durations, charges, prices, etc. and rests of descriptive information are stored in various dimensional tables. For example, in zone dimension, the call's destination details, such as country, city, district, and zone prefix are stored to describe the call destinations. Compared with the fact table, the numbers of tuples in the dimensional tables are relatively very small. But the size of attributes are much larger that fact tables. As fact table only contains keys and a few numeric attributes, the total size of the table is much smaller than a table to directly store data loaded from log files.

The model helps to increase query performance by reducing the volume of data that is read from disk. Queries analyse data in the dimensional tables to obtain the dimension keys which join the index in the central fact table to further reduce the number of tuples to be scanned.

5.1.3 Needs for More Efficient Join Algorithms

With success of the business, the number of customers and call usages has increased dramatically. It caused a significant increasing in transaction data and customer queries in the data warehouse system. However, the network capacity and stability of the Internet based VPN has generated significant performance problems for traditional query processing techniques as data access becomes less predictable due to link congestion, load imbalances, and temporary outages. As traditional join algorithms need to wait all data source ready before join process starts, users might require to waiting longer for receiving first desired record. In the worst case, they might never get the answers. Company has to place a query limits to the current SQL Server based data warehouse system to allow the system running smoothly. The performance issues have caused constraints on decision making of best telecommunication routing selecting with two hours delay rather than the runtime as preferred.

In order to make I.T.S data warehouse successful in the distributed and dynamic environment, it is essential to have better join algorithms for the data warehouse working effective and efficient in such environment, since the join is one of the most costly and

frequently running operations in the data warehouse system.

To fully realise a new data warehouse system for the company is beyond our scope, however, the data sets and multidimensional data model of scenario could be used for our study and experiments to help us to gain better understanding and evaluation of the modern adaptive join algorithms.

5.2. Simulated Network Environments for Experiments

As discussed earlier in chapter 4, the modern adaptive join algorithms have noticeable advantages over the conventional join algorithms in distributed environment. However, it is difficult to measure or evaluate the performance of those join algorithms at real world environment since the distributed and dynamic environment, such as changing of network speed, available memory space, available CPU power, will affect on the performance these join algorithms.

To obtain a scientific and meaningful results for a comparative study on these join algorithms, a controlled data warehousing experimental environment is used, in which the network speed and memory availability as main factors affect on the comparison of join performance among the different types of join adaptive algorithm are identified and used. Other factors such as CPU power of the server are very important in terms of overall join performance, but it affects all join algorithms in a similar way. It is not included as it is insignificant in the comparison of join performance among the different types of join adaptive algorithm.

Simulated network environments instead of real data warehousing environment are used for the experiments in order to create a repeatable environments to measure the performance of different join algorithms fairly.

In our study, four types of join algorithm: nested-loop ripple join, Hash Ripple Join, XJoin and XJoin without second stage (XJoin-No2) have been investigated. These join algorithm are implemented in Java based on the algorithms described in the chapter 4.

In the experiments, two relational files containing up to 100,000 tuples extracted from telecom data were used. One relational file is called customer.txt described as relation R in the following discussion, which has a total of 2.2MB data in comma-separated values (CSV) format and contains two attributes, accountID and customerName where The primary key is accountID. Another relational file is called outgoing.txt described as relation S in the following discussion, which has a total of 7.7MB data in CSV format and contains seven attributes with accountID as foreign key. For experimental purpose, all tuples for both relations have been random ordered.

Because performing experiments directly on a real world network would not provide repeatable results, four groups of data sets have been designed to model following four types of network behaviours.

- *Random* *20-120Kbytes/sec*
- *High Speed* *265-512Kbytes/sec*
- *Low Speed* *10-20Kbytes/sec*
- *Bursty* *20-120Kbytes/sec with 1 sec delay per 1000 tuples*

In the experiment different transfer rates were modelled by traffic delay. Because the record size for two relations are about 26bytes and 144bytes respectively, accordingly the average delay could be 1.3ms (26/20) and 7.2 ms (144/20) for 20KBytes/sec transfer rates. In the same way, 0.22 and 1.2 ms delays could be used to model 120 KBytes/sec transfer rates, 0.10 and 0.56 ms delays for 256KBytes/sec transfer rates, 0.05 and 0.28 ms delays for 512KBytes/sec transfer rates and 2.6 and 14.4 ms delays for 10KBytes/sec transfer rates.

In random model (Figure 5.4, and 5.5), network delay was generated randomly ranging between 0.22 ms and 1.3 ms for R relation and ranging between 1.2 ms and 7.2 ms for S relation. Total delay is about 72 sec for R and 420 sec for S.

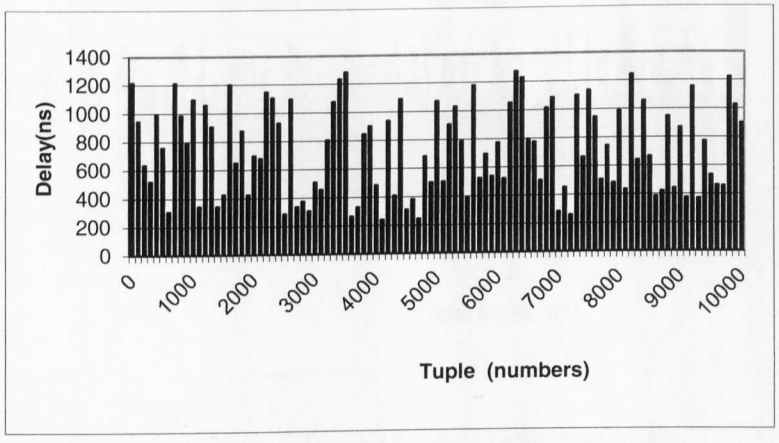

Figure 5.4 Random model for first 10000 tuples of R

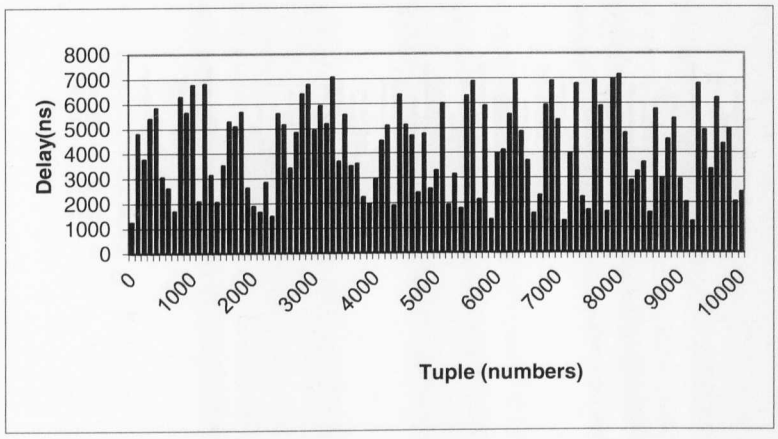

Figure 5.5 Random model for first 10000 tuples of S

In high speed model (Figure 5.6, and 5.7), network delay was generated randomly ranged between 0.05 ms and 0.1 ms for customer relation and ranged between 0.28 ms and 0.56 ms for outgoing relation. Total delay is about 7 sec for R and 41 sec for S.

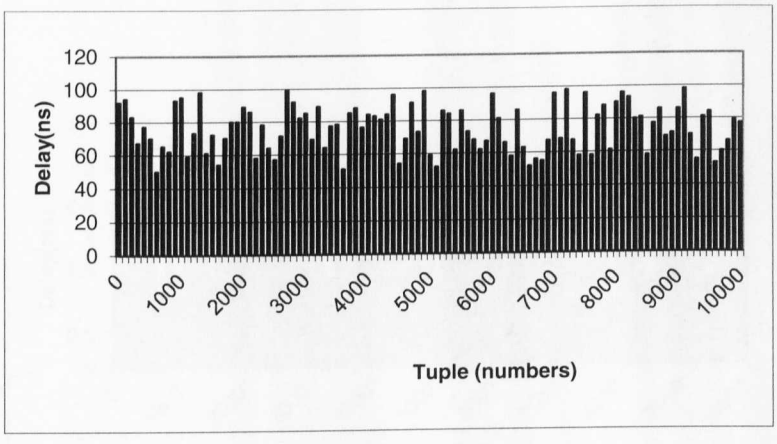

Figure 5.6 High speed model for first 10000 tuples of R

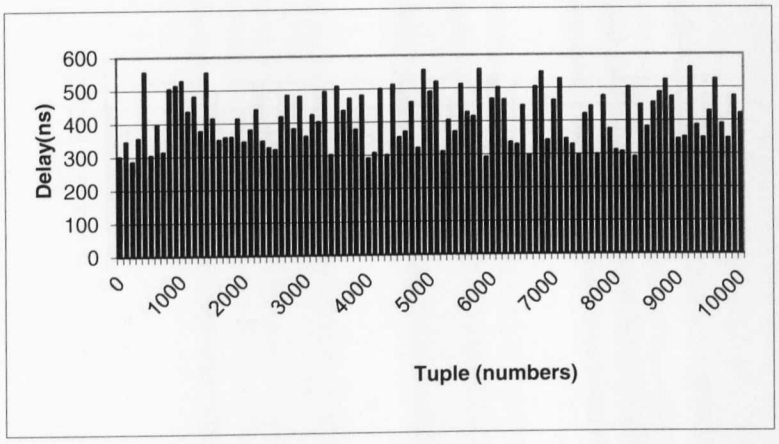

Figure 5.7 High speed model for first 10000 tuples of S

In low speed model (Figure 5.8, and 5.9), network delay was generated randomly ranged between 1.3 ms and 2.6 ms for customer relation and ranged between 7.2 ms and 14.4 ms for outgoing relation. Total delay is about 194 sec for R and 1080 sec for S.

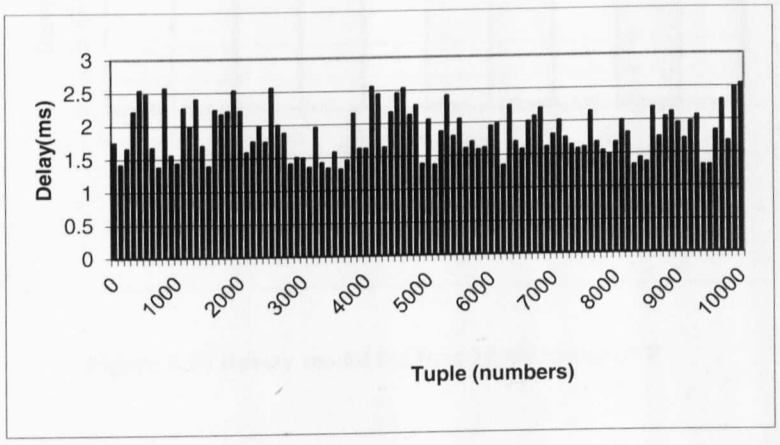

Figure 5.8 Low speed model for first 10000 tuples of R

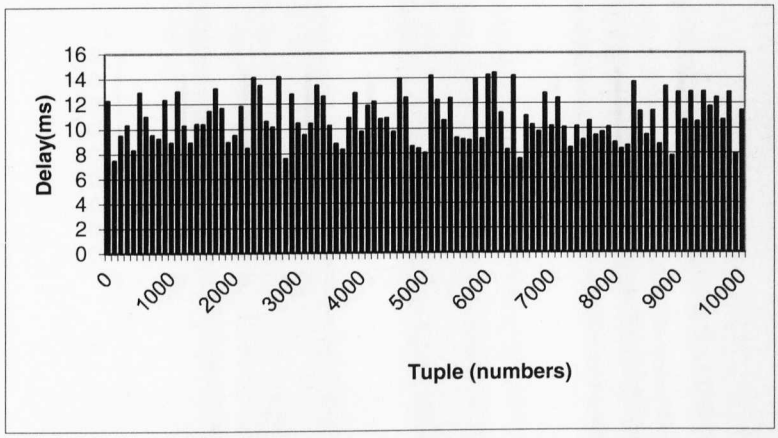

Figure 5.9 Low speed model for tirst 10000 tuples of S

In bursty model (Figure 5.10 and 5.11), extra 26 ms and 144 ms delay for R and S per 1000 tuples will be added to the random model. Total delay is about 74 sec for R and 434 sec for S.

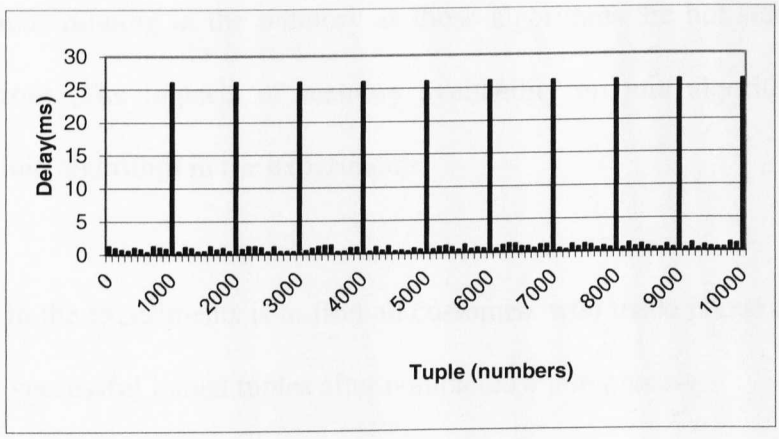

Figure 5.10 Bursty model for first 10000 tuples of R

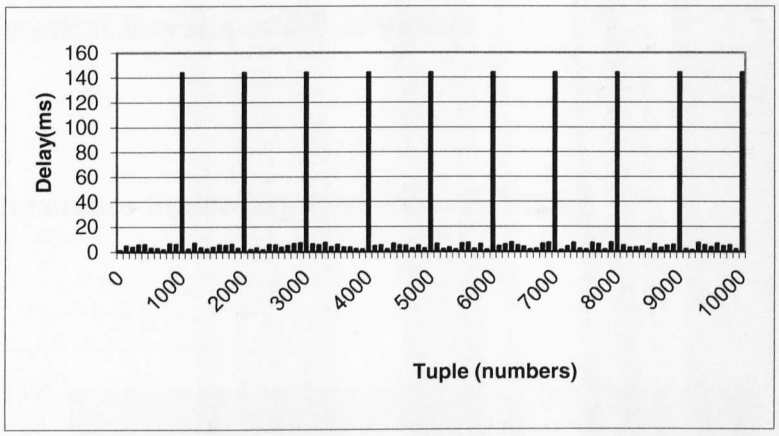

Figure 5.11 Bursty model for first 10000 tuples of S

The experiments were run on P4/2.8GHz window2000 platform, with 1GB of memory and 80GB of disk space. In the experiments, Standard memory for Xjoin was set to 5MB while Nest Loop Ripple Join and Hash Ripple Join were allocated enough memory to keep both relations running in the memory as those algorithms are not suitable for low memory conditions. The impacts of memory availability on join algorithms are also studied using Xjoin algorithm in the experiments.

The query used in the experiments is to find all customers who made phone call to China. It generates 247 successful joined tuples after completed a join process.

5.3. Experimental Results and Evaluation

5.3.1 Join Performance Under Random Network Model

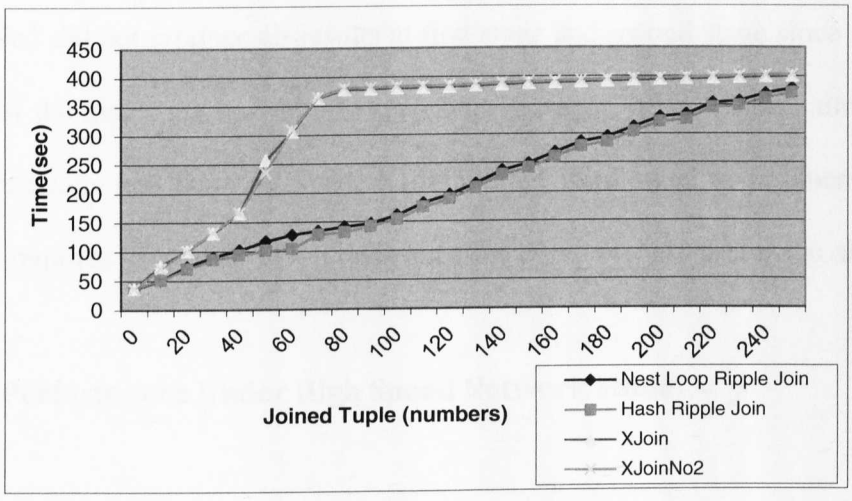

Figure 5.12 Join performance under random network

In the first set of experiments, the performance of Nest-Loop ripple, Hash ripple, XJoin and XJoinNo2 under random network model is examined. Figure 5.12 shows the cumulative response times for four algorithms. The x-axis shows a count of the results tuples produced and the y-axis shows the time as which that result tuple was produced.

The result shows that there is only little difference between Hash ripple and Nest-Loop ripple join and between XJoin and XJoinNo2. Because network I/O is much slower than memory or disk operation in random network model, although Nest-loop ripple join required much more memory operation than Hash Ripple Join did, it did not make big differences between them in the overall performance. Second stage of XJoin has very

little help for improving the performance because there are not very long delays for second stage to do work in the gap to improve XJoin performance.

However, XJoin and XJoinNo2 become slower than Hash ripple and Nest-Loop ripple join after receiving 20 tuples, but they speed up after matched 80 tuples. It is due to XJoin and XJoinNo2 did not produce all results at first stage and second stage since some of the hash table of the tuples are moved into secondary storage. After 375sec, almost all data from two relations had been received, XJoin started third stage to produce all results. Although it requires some disk I/O, it does not slow down the join process at all.

5.3.2 Join Performance Under High Speed Network Model

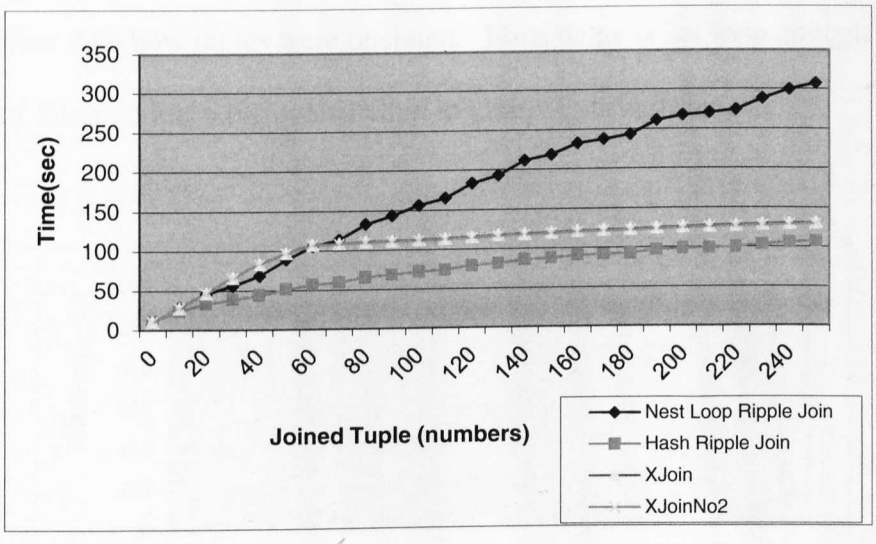

Figure 5.13 Join performance under high speed network

In the second set of experiments, the performance of Nest-Loop ripple, Hash ripple, XJoin and XJoinNo2 under high speed network model is examined. The result in Figure 5.13 shows that Hash Ripple Join gained best performance, while Nest-Loop ripple join got worst performance. It is due to network delay becomes less important factor in the join

environment and the time used by join operation becomes noticeable. Because XJoin requires some disk I/O and do not produce all join results at first and second stage, the performance is inferior to Hash Ripple Join. However, XJoin is superior to Nest-loop join because hash method has been applied in XJoin algorithm. Due to same reason as random network model, there is hardly difference between XJoin and XJoinNo2.

5.3.3 Join Performance Under Low Speed Network Model

In the third set of experiments, the performance of Nest-Loop ripple, Hash ripple, XJoin and XJoinNo2 under low speed network model is examined. The result in Figure 5.14 shows that the performance among four join algorithms is similar to that under random network model. It proves that the join performance was affected by how tuples were received rather than how tuples were operated. 14ms delay is not long enough for second join stage of XJoin to make big contribution to improve its performance.

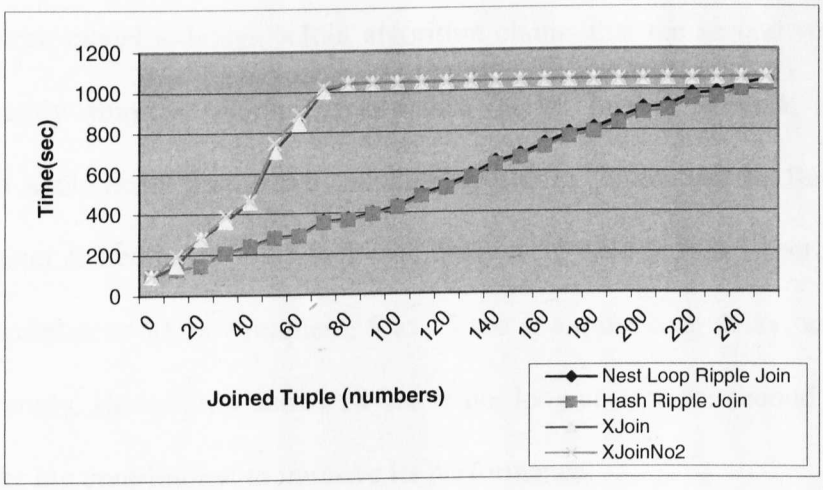

Figure 5.14 Join performance under low speed network

5.3.4 Join Performance Under Bursty Network Model

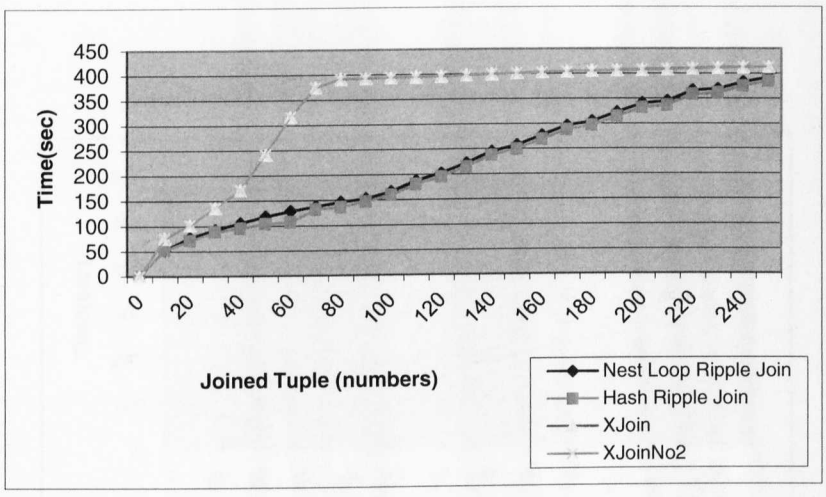

Figure 5.15 Join performance under bursty network

In the fourth set of experiments, the performance of Nest-Loop ripple, Hash ripple, XJoin and XJoinNo2 under bursty network model is examined. Surprisingly, the result in Figure 5.15 shows that the performance among four join algorithms is still similar to that under random network model although XJoin algorithm claims that the second stage of XJoin can significantly improve XJoin performance under bursty network environment. According to XJoin result provided by Justin Forrester in "Xjoin and the Benefits of free work" (Forrester & Ledlie, 2002), 5 to 15 sec delay were used in their experiments. Since in the high speed network environment, 5 to 15 sec is a quite long delay, we use 144ms delay in our study. However, it shows 144ms is not long enough for second join stage of XJoin to make big contribution to improve its performance.

In order to further investigate the XJoin algorithm, the fifth set of experiments with a new bursty network model was designed which apply 2 sec delay to high speed model

described in section 1.2. The new result is showing in Figure 5.16. It shows that XJoin performance can be improved dramatically from XJoinNo2 under longer delay bursty network. The performance is near to Hash Ripple Join which requires full memory support.

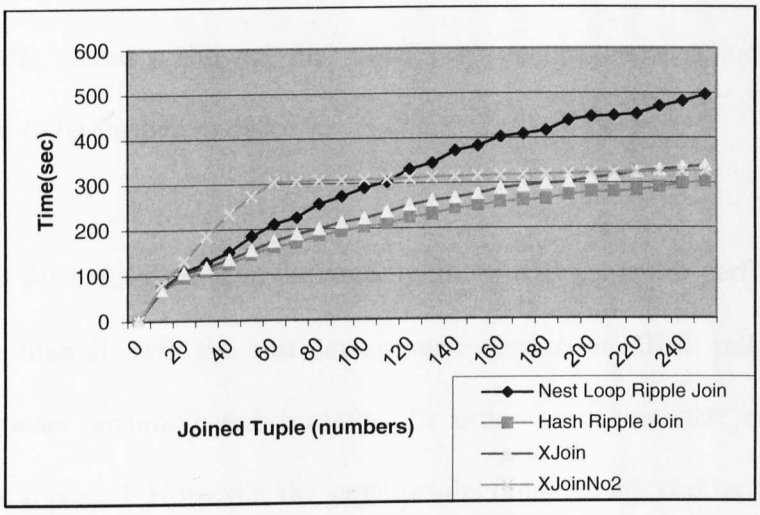

Figure 5.16 Join performance under bursty (2 sec delay) network

5.3.5 Join Performance Using XJoin with different memory size

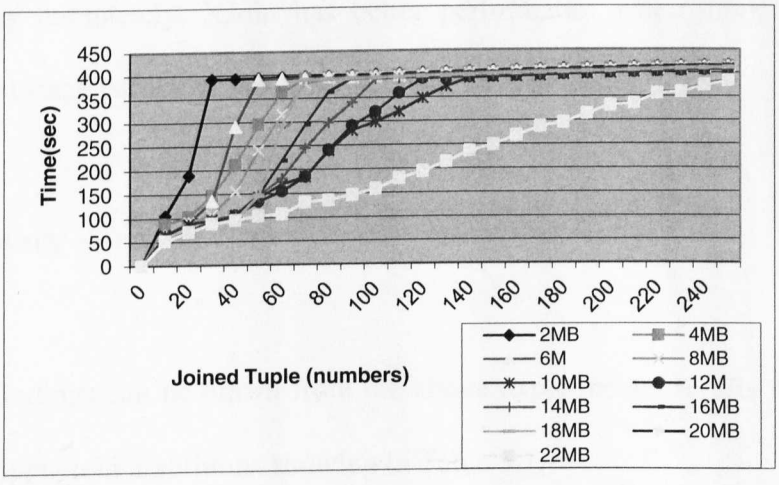

Figure 5.17 XJoin performance with different memory sizes

In the sixth set of experiments, the performance of XJoin with different memory size under random network model is examined. The results from experiments showed in Figure 5.17 indicated that XJoin would obtain best performance when more than 20MB memory allocated, which mean the whole join operation could be run in the memory. XJoin with 2MB memory showed the worst performance because most of the hash partition needed to be flushed to disk.

Unexpectedly it does not show that the more memory will get better performance. When memory is less than 20 MB, the best performance appears at 10MB rather than 18MB. The result appears unanticipated initially. In order to insure the correctness, the experiments are repeated. However, the same results obtained are similar to the previous one. After analysing XJoin algorithm in depth, it can be found that the more memory allows more tuples be kept in the memory. When the hash partition needed to be flushed to disk, it actually slows down the join process because it requires larger partition to be flushed to disk. It explains why the more memory available may not be necessary to have better performance if the memory space is not sufficient to keep the whole hash partition in the memory completely. XJoin has better performance when more tuples can be matched in first stage particular in slower network environments.

5.4. Summary

Some useful findings can be drawn from the above experimental results to compare the four types of ripple join algorithms showing in Table 5.1.

	Nested-Loop Ripple Join	Hash Ripple Join	Xjoin	Xjoin without second stage
Basic Algorithm	Nested Loop	Hash	Hash + secondaray storage	Hash + secondaray storage
Opeartion Stages	1	1	3	2
Backgroud Process	N/A	N/A	Join mem partition with disk partition	N/A
Memory required	Fully	Fully	As little as 2MB	As little as 2MB
High speed network	Poor	Good	OK	OK
Brusty network (long delay)	Poor	Good	Good	OK
Other network	Very Good	Good	OK	OK
Advantages	Used for any join	Best performance	Low memory requirements	Low memory requirements
	simple		Taking advantage of delay	
Main problems	Too slow in high speed network	Equijoin queries	Equijoin queries	Equijoin queries
	High memory comsumption	High memory comsumption		
Improvement probability	Index, block			

Table 5.1 Comparison of ripple join algorithms

The Hash Ripple Join algorithm has provided the best performance among the join algorithms, but it requires enough main memory to store the whole relations to be joined. Because the data stored in a data warehouse typically represents data over a long period of time, it might be impossible to provide such an environment to support the join algorithm.

The XJoin algorithm utilising secondary storage successfully reduced the memory requirements of the join algorithm. However, join performance was affected noticeably at first stage of the join because of the join results might be delayed to produce when some tuples were moved to secondary storage.

The second stage of the XJoin may be helpful to reduce the delay. However, the algorithm adapting to the changing of data environments are based on behaviours of data itself rather than the reason behind the behaviours. For example, to decide whether the second

stage of the XJoin should be started was based how long had been delayed to receive the tuples rather than how long would be needed to receive the tuples or what is the cause of the delay. Therefore, selecting and scheduling of join algorithms and stages might not be wise, which could lead join operation to longer delay or overheated by swapping among the different join stages.

The experimental results have indicated that different type of join algorithm should be selected according to the different kinds of environments. Hash Ripple Join is most preferable join algorithm for all environments if enough memory is available. XJoin is suitable for lower memory available environments and particular in long delay bursty network environment.

These pipelined join algorithms offer effective approaches to deal with the unpredictability of distributed and dynamic data environment. Experimental results have enhanced our confidence to address the issues arisen from distributed and dynamic data warehouse environments. However, these algorithms have not optimized for current data warehouse environment where the size of a fact table is usually extremely large and most of keys are heavily indexed. Another inadequacy of these algorithms is that their adaptiveness to the changing data environments is based on behaviour of data itself rather than the reason behind the behaviour. Therefore, selecting and scheduling of join algorithms and stages might not be 'optimised', resulting in longer delays and 'overheating' in join operations. An intelligent and specialised algorithm is needed for the distributed and dynamic data warehouse environment.

Chapter 6. AJOIN FRAMEWORK AND AJOIN ALGORITHM

This chapter presents a proposed agent-based join algorithm called AJoin for effective and efficient online join operations in distributed data warehouses. Taking into consideration data warehouse features, AJoin utilises intelligent agents for dynamic optimisation and coordination of join processing at run time. The algorithm extends both modern adaptive join and semi-join techniques. As a result, the AJoin improves join performance significantly at various network conditions.

Compared with the traditional join algorithm, modern ripple join algorithms demonstrate better adaptivity to the changing environment. However, these modern algorithms were designed for the general join purpose and did not take distributed data warehousing features into account. A typical dimensional data warehouse model consists of fact tables connected with many dimensional tables which contain descriptive textual information (Kimball & Ross, 2002). In a dimensional table, the size of join attributes is usually much smaller than the size of the whole attribute set. Furthermore, only small numbers of tuples in the dimensional table participate in the join with the fact table at run time.

Software agents have been used in a wide variety of applications involving distributed computation, communication, or autonomous operations (N. R. Jennings, 2001). In the AJoin algorithm, a join task is decomposed into smaller independent sub-tasks to be assigned to software agents. For example, a Remote Information Agent (RIA) could be applied to filter unqualified tuples in a remote site before sending required data back so that the amount of data transition could be reduced. All agents are working continuously and independently in a multi-agent system where they are coordinated according to its

environment states (such as network speed, memory availability). The environment states are closely monitored and used as feedback to enable agents to decide how to adapt their behaviour to the changing environment at runtime.

Semi-join as a traditional join method for reducing data transmission in processing distributed queries was introduced in (Bernstein & Chiu, 1981)'s project. This method is especially beneficial in a wide-area distributed network environment such as the VPN where the cost of data transmission is usually much higher than local processing costs. AJoin adopts the principles of the semi-join to transmit only the join attributes rather than all attributes from relations to start a join. In addition, AJoin enhances the conventional semi-join approach by eliminating unqualified tuples at remote sites to minimise the transmission cost. Furthermore, as semi-join requires multiple scans of relations, it may not be as cost effective in a high speed networking environment. To address this problem, AJoin utilise software agents to dynamically adapt join between full-join and semi-join at runtime based on a function involving network speed. As a result, the AJoin improves join performance significantly at various network conditions.

Similar to modern adaptive join, AJoin divides the join task into a series of sub-tasks for parallel processing in a ripple manner. This enabled join results to be produced as soon as join tuples are matched. Join processing continues even if when one of data sources is temporary unavailable.

The rest of this chapter is organised as follows. Section 6.1defines some terms used in the discussion of the AJoin framework and its algorithm, Section 6.2 presented the proposed AJoin framework and explains the process in details. Section 6.3 describes the AJoin

algorithm in Java-style pseudo code. Section 6.4 provides a cost-benefit analysis for the AJoin and explains the development of adaptive join strategy.

6.1. Definition

The AJoin algorithm can be extended to process multiple relations, for simplicity reasons our discussion in the following sections considers only two relations, R1 and R2, which reside at remote sites S1 and S2 respectively. S denotes a local site where join results are produced.

Definition 1 (Join Attributes) Join attributes are fields in a relation used for matching tuples with another relation for a join operation. R_A denotes join attributes in relation R.

Definition 2 (Selected Attributes) Selected attributes denoted as R_S, are those attributes which are required as part of the output from join operation.

Definition 3 (Attribute Size) Attribute size is the size of the related attributes in bytes. $Size(R_A)$ denotes the size of join attributes, $Size(R_S)$ denotes the size of selected attributes, and Size(R) denotes the total size of all attributes.

Definition 4 (Additional Query Conditions) Additional query conditions, denoted as C, is a set of further conditions (other than the join condition itself) required for a query.

Definition 5 (Join Cardinality) Join cardinality JC denotes the number of tuples in the joined relation. Join cardinality of relation R denoted as JC(R) is the number of unique tuples from relation R which are required to produce joined tuples.

6.2. AJoin Framework

A framework of AJoin is illustrated in Figure 6.1. AJoin is operated in a multi-agent system where the join task is decomposed into smaller independent sub-tasks carried out by different agents and coordinated by another specific agent called the Join Coordinator Agent (JCA). AJoin processing can be divided into four phases: Initialization, Remote Adapting, Ripple-adaptive Join and Result Output.

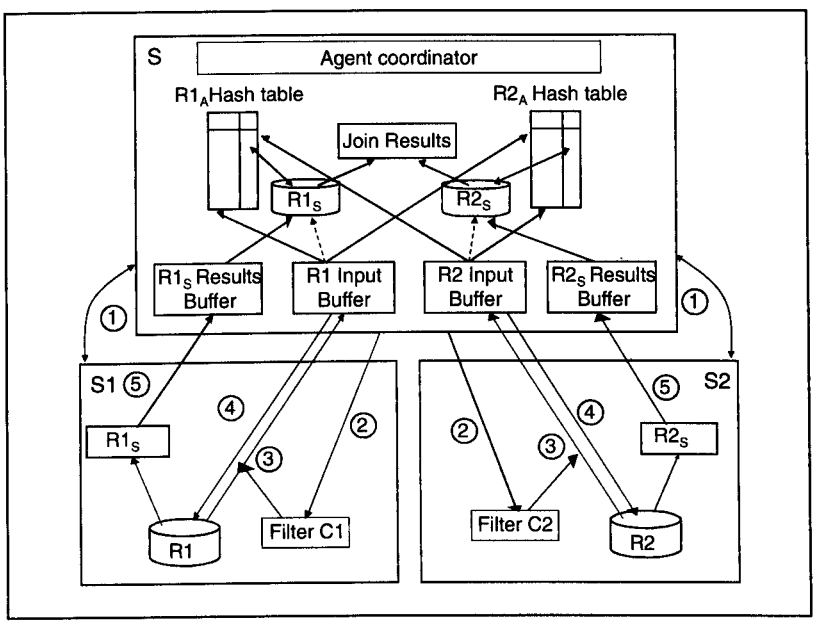

Figure 6.1 AJoin framework

In the first phase (Initialization), the JCA collects local and remote environment states, as shown by ① in Figure 6.1, including available memory, network connection speed and information on join relations. Using that information the JCA is able to allocate initial memory for join hash tables and local storages for both join relations. It can also perform

a cost-benefit analysis of the join to calculate the join switching threshold (to be discussed in section D) which can be used at runtime to select a join method.

The second phase (Remote Adapting) is shown by ② in Figure 6.1. Additional query conditions C1 (and C2) and join switching threshold are sent to remote sites S1 (and S2) where Remote Information Agents (RIA) are activated and start to produce tuples from R1 and R2 using the conditions C1 and C2 respectively to filter unqualified tuples. Once qualified tuples are retrieved, the agent will choose a join method based on the cost-benefit analysis involving network speed.

If full-join method is chosen, both R_A and R_S of relation R will be sent back to the Input Buffer (IB) at the local site S illustrated as ③ in Figure 1. Otherwise, only R_A and its remote direct access address, called Remote Access Pointer (RAP), will be sent back to the IB. The RIA will continue its operation and select the join method to adapt to changing network speed until a request is received to end the process.

The third phase (Ripple-adaptive Join) is the main join phase, in which the join operation is dynamically switched between full-join and semi-join algorithm according to the selection decision made by RIA at phase 2.

Once an IB starts to receive data from a remote site, Tuple Matching Agent (TMA) at site S activates the ripple-adaptive join process. Two hash tables for R1A and R2A are used for a tuple matching process. In the hash table for R_A, the R_S access pointer (used to locate R_S which could be at local or remote site) is stored and sorted on R_A.

For any newly received tuple, the value of its access pointer is based on data contained in the IB. When both R_A and R_S are received, which indicates the full-join method is selected, RS will be sent to the local storage, shown as a dashed line in Figure 1, and the Local Access Pointer (LAP) of R_S will be add to the hash table for R_A.

If a received tuple only contains R_A and its RAP, it means that the semi-join is chosen. In this case, the value of the RAP will be set to negative and then saved into the hash table for R_A together with R_A itself. The negative value of the access pointer indicates that the RS still remains at the remote site.

In the meantime, the TMA will use $R1_A$ (or $R2_A$) to probe $R2_A$ (or $R1_A$) in the hash table for $R2_A$ (or $R1_A$) to find a matching tuple. If a match is found, the TMA calls an Output Agent (OA) to produce the join results and then continues its ripple-adaptive join processing until all the tuples at the IB have been processed.

The final phase (Result Output) retrieves the matched tuples and produces the join output. The first task for the OA is to check the access pointer to determine whether the required attributes have been stored in the local storage at site S. If the value of the access pointer is positive, it means that required R_S is directly accessible from the local storage and the join results can be produced.

On the other hand, if the value is negative, it means that the required R_S need to be retrieved from a remote site. In this case, the RIA will be requested to retrieve R_S shown by ④ in Figure 1. At the remote site, the RIA will retrieve required Rs based on RAP and then send it back to the local site S illustrated by ⑤ in Figure 1. At the local site S, the

OA stores the Rs in the local storage, updates the access pointer values in the hash table for R_A and produce the join results.

6.3. AJoin Algorithm

Join coordinate at local site S:

1. Initialization.
   Perform cost-benefit analysis;
   Allocate initial memory and buffer.
2. Activate RIA at sites S1 and S2 (see table 6.4)
   Start Remote Adapting
3. Activate MTA and OA at site S (see table 6.2)
4. Coordinate and monitor join process
5. Iterate from step 4 until join completed
6. Deactivate MTA, RIA, and OA
7. End AJoin

Table 6.1 Join coordinator agent (JCA)

MTA for R1 at local site S:

1. Receive a tuple from IB (see table 6.3)
2. If a tuple is received from R1
3. If full-join method is selected
4. Save $R1_S$ at local site S and add its LAP and $R1_A$ to hash table for $R1_A$
5. Else // semi-join is selected
6. Add negative RAP and $R1_A$ to hash table for $R1_A$
7. Using R1A to probe $R2_A$ in hash table for $R2_A$
8. If a match is found
9. For each matched $R1_A$ and $R2_A${
10. If the access pointer of R_S is RAP
11. Call RIA at S1 and/or S2 (see table 5) to retrieve $R1_S$ and/or $R2_S$ respectively
12. Else // R_S accessible from the local storage
13. Retrieve $R1_S$ and/or $R2_S$ locally
14. Output the matched $R1_S$ and $R2_S$
15 }
16. Iterate from step 1 until no more tuples from R1
17. Notify JCA to end the process

Table 6.2 Tuple matching agent (MTA)

Similar algorithm to MTA for R1, MTA for R2 operates ripple-adaptive join processing for R2 until all the tuples at the R2 IB have been processed.

Receive tuple procedure at local site S:

```
1. If all tuples from both relation R1 and R2 are received
2.      Join completed
3. If both R1 and R2 buffer are not empty
4.      Retrieve a tuple from R1 and R2 IB in turn
5. Else
6.      If both R1 and R2 IB are empty
7.              Waiting tuples to arrive
8.      Else    // one of R1 orR2 IB is not empty
9.              If R1 IB is not empty
10.                     Retrieve a tuple from R1 IB
11.             Else // R2 IB is not empty
12.                     Retrieve a tuple from R2 IB
13. Return
```

Table 6.3 Receive a tuple procedure

Tuple filter and join adapting at remote site S1:

1. Receive join parameter (C and join selection threshold)
2. Retrieve a tuple from R1
3. If the tuple is not meet the condition C1
4. Repeat step 2
5. Evaluate join cost-benefit
6. If semi-join is selected
7. Send R1A and its remote access pointer to R1 IB at site S
8. Else // full-join is selected
9. Send R1A and R1s to R1 IB at site S
10. Iterate from step 2 until no more tuples from R1

Table 6.4 Remote information agent (RIA)

Similar algorithm to table 6.5 is used for RIA at remote site S2 for R2.

Retrieve R_S at remote site S1 (or S2):

1. Get the remote access pointer
2. Retrieve the tuple according to its access pointer
3. Return R_S to R_S buffer at site S

Table 6.5 RIA retrieve RS service

6.4. Cost-benefit Analysis

Being one of the most expensive operations in query process, the cost of the join operation has been investigated intensively in the previous work (L. M. Haas et al., 1993), (Harris & Ramamohanarao, 1996), (Li et al., 2007). In a low bandwidth distributed network environment, compared the cost of local processing with the cost of data transmission, the cost of local processing is relatively small and could be ignored.

Given |R1| and |R2| to indicate the cardinality of R1 and R2 respectively, on the one hand, the cost of full ripple join is specified as below:

$$\text{Cost}_F(R1 \bowtie_A R2) = T_0 + \frac{|R1| \times Size(R1) + |R2| \times Size(R2)}{V} \quad (1)$$

where T_0 is the cost to start-up a new network connection and V is the network speed.
On the other hand, in the semi ripple join method, $R1_A$ and $R2_A$ from remote sites S1 and S2 are retrieved for matching, and only R1s and R2s of matched tuples are then further retrieved from sites S1 and S2, Therefore, the cost of the semi ripple join can be expressed as below:

$$\text{Cost}_S(R1 \bowtie_A R2) = T_0 + \frac{|R1| \times Size(R1_A) + JC(R1) \times (Size(R1_S) + Size(P))}{V} + \frac{|R1| \times Size(R2_A) + JC(R2) \times (Size(R2_S) + Size(P))}{V} \quad (2)$$

where Size(P) is the size in byte of the RAP.

The cost and benefit of AJoin between the two join methods can be measured in their differences as below:

$$\text{Cost}_F(R1 \bowtie_A R2) - \text{Cost}_S(R1 \bowtie_A R2) =$$

$$\frac{|R1| \times Size(R1) - |R1| \times Size(R1_A) - JC(R1) \times (Size(R1_S) + Size(P))}{V}$$

$$+ \frac{|R2| \times Size(R2) - |R1| \times Size(R2_A) - JC(R2) \times (Size(R2_S) + Size(P))}{V} \quad (3)$$

In AJoin, full-join or semi-join can be applied independently to each relation. The semi-join method will be chosen only when the following condition holds for a relation:

$$\frac{|R| \times Size(R) - |R| \times Size(R_A) - JC(R) \times (Size(R_S) + Size(P))}{V} > 0 \quad (4)$$

or

$$CR(R) \times SR(R) < 1 \quad (5)$$

where

$$CR(R) = \frac{JC(R)}{|R|} \text{ denotes as join cardinality ratio as \%;} \quad (6)$$

$$SR(R) = \frac{Size(R_S) + Size(P)}{Size(R) - Size(R_A)} \text{ denotes as attribute selection ratio as \%} \quad (7)$$

The results show that the network speed is no longer a consideration factor for choosing join methods in a low bandwidth network condition. CR(R) and SR(R) will be the factors to determine which join methods to be used.

A join in ad-hoc queries in a typical data warehouse could be divided into multiple one-to-many joins between the fact table and the dimensional tables. When referential integrity constraints are applied between the fact table (R1) and the dimensional tables (R2), all join attributes in fact table must have a matching join attributes in the dimensional tables. Therefore, CR(R1) will be 100%. In such cases, a full-join method will be used for R1 unless SR(R1) is low.

However, since run-time data of the fact table only represents a small portion of the fact table, CR(R2) will be normally far less than 100%. Therefore, the semi-join method should be chosen for R2.

There have been significant improvements of Internet bandwidth recently, with 45Mbps T3 internet connection already available for business users and higher bandwidth connection on the way. In these high bandwidth networks, the join cost of local processing compared with the cost of data transmission becomes more significant and cannot be ignored any more.

The semi-join method involves more local processing than the full-join method. The cost of additional local processing may outstrip the benefit of the semi ripple join method when the network speed has increased to a certain level. To evaluate the true cost of a join is not a trivial exercise due to many factors which needs to be considered (L. M. Haas et

al., 1993). However, as the cost of joins at a local site using full or semi-join method is at an equivalent level, we only need to calculate the difference between the two methods when remote site are involved.

The equation (4) could be revised into the following to decide which join method should be chosen:

$$\frac{|R|\times Size(R)-|R|\times Size(R_A)-JC(R)\times(Size(R_S)+Size(P))}{V} > \frac{JC(R)\times Size(R_S)}{V'} \quad (8)$$

where V' is the desk data access speed. Since the JC(R), |R|, Size(R), Size(R_A), Size(R_S) and Size(P) all could be queried before AJoin starts, the join switch threshold of V could be pre-calculated as below:

$$V' \times \frac{|R|\times Size(R)-|R|\times Size(R_A)-JC(R)\times(Size(R_S)+Size(P))}{JC(R)\times Size(R_S)} \quad (9)$$

This avoids the needs for such calculation at runtime when selecting a join method at runtime.

The threshold above indicates a turning point where the cost of additional local processing outweighs the benefit of the semi ripple join method

Chapter 7. EVALUATION

This chapter presents a performance study to evaluate the effectiveness of the AJoin algorithm and discuss the outcome of this study. The evaluation is based on a comparative study of AJoin against other modern join algorithms using the following three assessment matrix: network speed, memory consumption, and join speed. The overall evaluation results have demonstrated that AJoin has consistently outperformed the other modern join algorithms. In the slower network setup, AJoin performs particularly well with a performance improvement of an average of 30%-54% against Hash Ripple Join.

All algorithms used for evaluation are implemented in Java and evaluated in the controlled network simulation environment. Cost effective VPN network based on Asymmetric Digital Subscriber Line (ADSL) and Fibre Optical connections, leased T1 based Wide Area Network (WAN) connection, as well as Gigabit High speed network connection have been used to test the join algorithms.

The rest of this chapter is structured as follows. Section 7.1 describes the evaluation environment and its setup. It justifies the use of controlled experiment environment for the evaluation. It explains why network speed and memory availability are selected as key factors for the evaluation. It also gives reasons for choosing Hash Ripple Join for comparative study under sufficient memory environment and XJoin for comparative study under variable memory conditions. Environment setup including join relations and network speed are reported. Section 7.2 presents the performance of AJoin with sufficient memory environments. Evaluation results of AJoin against Hash Ripple Join under four

types of simulated network environments with four set of queries are reported. To further study the effectiveness of the adaptive function of AJoin, evaluation results of AJoin performance without remote filtering function are also reported. To evaluate the AJoin performance under bursty network condition, evaluation results of AJoin performance against Hash Ripple Join are presented. Section 7.3 presents the performance of AJoin with limited memory environments. Evaluation results of AJoin against XJoin under four types of simulated network environments with four set of memory settings are reported. To evaluate the AJoin performance under bursty network condition, evaluation results of AJoin performance against XJoin are also presented. Section 7.4 provides a summary of the evaluation results. Based on the evaluation outcomes, it concludes that AJoin has great scalability and adaptivity. It works perfectly in various network conditions and memory spaces. It automatically adapts itself to the changing environment to achieve possible best performance. AJoin has consistently outperformed the Hash Ripple Join algorithm and XJoin. In the slower network setup, AJoin performs particularly well and it improves performance against Hash Ripple Join by an average of 30%-60%.

7.1. Evaluation Environment and Its Setup

According to the theoretical study on the costs and benefits analysis in chapter 6, the AJoin could be beneficial in distributed data warehousing environments. The benefits are in inverse proportion to join cardinality ratio and attribute selection ratio described in formula (7) in chapter 6.4. The benefits may vary when local processing costs are taken into account. According to equation (8) in chapter 6.4, the benefits are also in inverse proportion to network speed and local processing costs.

7.1.1 Evaluation Environment

In order to verify our theoretical research outcomes and evaluate the effectiveness and performance of AJoin, the following evaluation approach is adopted.

- **Controlled experiment environment**

In an uncontrolled distributed and dynamic environment, it is very difficult to produce comparable performance results of join algorithms for measure or evaluation and very challenging to determine the main factors which may have affected the join performance. Therefore, in order to obtain scientific and meaningful results for the comparative study on AJoin against other join algorithms, a controlled data warehousing experimental environment is used.

Since the controlled environment is simulated, some factors in the real network are not well modelled such as work latency, protocol overhead, and driver efficiency. The patterns of network models are far more diverse than simulated network models. As results, the real network speed will be usually much slower than simulated network models. Since AJoin are in general able to gain more benefit from slower networks based on our theory study, AJoin will perform even better in the real network environment against other modern join algorithms if AJoin can perform better in the simulated network models.

- **Network speed and memory availability as key factors for evaluation**

Join performance could be influenced by multiple factors. Some facts such as CPU process power are very important in terms of overall join performance, but it affects all join algorithms in a similar way. Consequently it is not included in this study as it is insignificant in the comparison of join performance among different types of join algorithm.

Hard disk access speed is also an important factor for joins. For the same reason as CPU process power, it has not been assessed in this evaluation. Some discussions have been included when join algorithms are assessed in a Gigabit network environment where the local process costs could not be ignored.

In this comparative evaluation, network speed and memory availability are used as main factors for the comparison of AJoin performance with chosen modern join algorithms.

- **Hash Ripple Join for sufficient memory environment**

Based on the outcomes of the comparative study on modern join algorithms discussed in chapter 5, Hash Ripple Join has exhibited the best performance when the available memory is sufficient to hold both hash tables for two relations completely. Therefore, the Hash Ripple Join has been chosen as a benchmark for the performance study to evaluate AJoin in a memory-sufficient environment where no hash table overflow will occur.

- **XJoin for variable memory conditions**

To assess the effectiveness and performance of AJoin in a limited memory environment, XJoin will be used as a benchmark for the performance study since XJoin is the most flexible modem join algorithm and capable of working effectively in variable memory size conditions. Since the memory requirement varies according to the size of hash table, to evaluate the performance, variable memory size conditions based on the 5%, 10%, 20%, and 50% percentage of required memory have been used.

7.1.2 Environment Setup

- **Join relation**

In the experiment setup, two relations are used and both relations contain up to 100,000 tuples extracted from a telecom data warehouse. One relation (R1) contains 524 bytes of customer account information, which includes AccountID, AccountName, CompanyName, ContactName, ContactTitle, Address, City, Region, Postcode, Country, Phone, Mobile, Fax, and Notes. For ethical and privacy reasons, only AccountID and AccountName are kept and the rest of details are simply called "customer details" in the data set. The AccountID is the primary key of the relation R1.

Another relation (R2) contains 276 bytes of call details, which include Call_ID, CallDataTime, DestinationPhoneNo, DestinationZoneName, CallEventMessage, CallProviderName, CallLineIdentification, and AccountID. The AccountID is the foreign

key of relation R2, which will be joined with the AccountID in the relation R1.

One of the queries to evaluate the join performance is to list all details of customers as well as call details of the account which is used to make phone calls to China. There are 247 matched tuples randomly distributed between the two relations.

- **Network speed**

To reflect the current WAN connection used for distributed data warehouse business applications as well as future development, VPN as one of the most popular cost effective options for WAN connection is used for network simulation. Gigabit network is not widely used as WAN connection option as because of high cost, but as it reflects the trends of the future development, it is also included in the simulation models to evaluate the join performance. The four network models are labelled as Low speed, Random speed, High speed, and Gigabit network. All four network models with bursty effects to model network temporarily unavailable will also be assessed. The details of the four 'network models' used in this study are:

- Low Speed VPN based on 2Mb ADSL connection
- Random Speed VPN based on 2-8Mb ADSL connection
- High Speed VPN based on 100M Fibre Optical connection
- Gigabit 1Gb network
- Bursty drop network speed to 1Kb/s per 1000 tuples

7.2. Performance of AJoin with Sufficient Memory Available

The development of computer technologies has made it increasingly possible for more memories to be available with lower cost for servers. In this section, we assume that there are sufficient memories available for join operations. There will not be hash table overflow occurring in Hash Ripple Join process. AJoin will be evaluated against Hash Ripple Join under four simulated network environments with four sets of queries.

7.2.1 AJoin Performance Under Low speed Network

The evaluation results shown in the following Figure 7.1 presents the join performance of AJoin and Hash Ripple Join algorithms under low speed network conditions. The following query Q1 is used for the evaluation and the query will output 247 match tuples.

```
Q1. List call details for customers who made phone call to China.
    Select * from R1, R2
    where R1. AccountID = R2. AccountID  and DestinationZoneName ='CHINA';
```

In Figure 7.1, the X axis represents the number of joined tuples and the Y axis represents the accumulated time used for each join in seconds. The blue line indicates the performance of AJoin and the red line indicates the performance of Hash Ripple Join. The evaluation results shown in Figure 7.1 demonstrate that AJoin is performing significantly better than Hash Ripple Join. AJoin successfully produced the first joined tuple in 4.5 seconds compared with 19.9 seconds using Hash Ripple Join. It improves the

performance significantly by 77.4%. The performance improvement is contributed by both remote filtering function and semi join methods selected by AJoin. AJoin has benefitted with mobile agents which filter the unqualified tuples before they are transmitted. AJoin has also benefitted by transmit only join attributes of the tuples for matching join tuples. Those not matched tuples will be no longer required to be transmitted. Since the join is running under low speed network condition, it will reduce the transmission cost significantly.

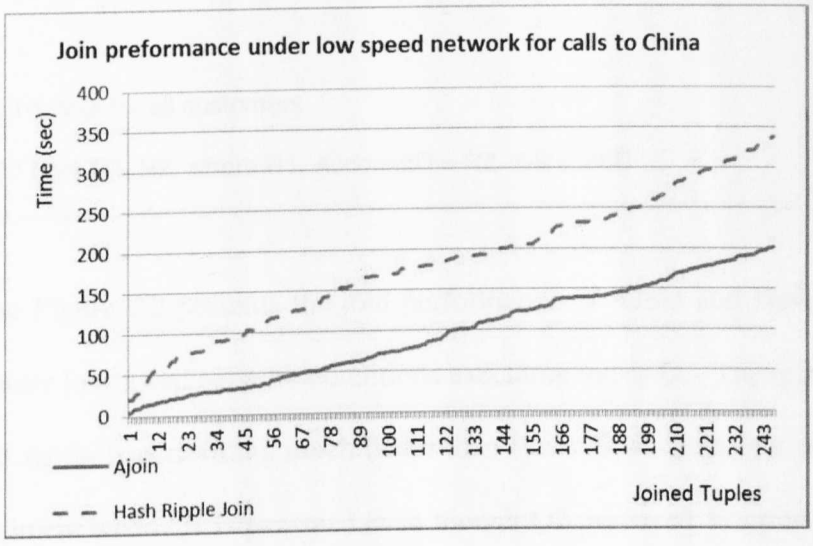

Figure 7.1 Join performance under low speed network for calls to China

As AJoin requires to transmit a joined tuple twice – join attributes of the tuple and selected attributes for output, it involves additional transmission cost, in particular, when more tuples will be matched. The benefits are not increased in a liner manner and typically more benefits at beginning of join stage. However, comparing with Hash Ripple Join, AJoin still improves the performance significantly by 51.8% on average.

To study the impact of the remote operation, the following queries are also used for the evaluation:

```
Q2. List call details for customers who made phone call to France.
    Select * from R1, R2
    where R1. AccountID = R2. AccountID and DestinationZoneName ='FRANCE';

Q3. List call details for customers who made phone call to USA
    Select * from R1, R2
    where R1. AccountID = R2. AccountID and DestinationZoneName ='USA';

Q4. List call details for all customers
    Select * from R1, R2  where R1. AccountID = R2. AccountID ;
```

The following Figure 7.2 presents the join performance of AJoin and Hash Ripple Join algorithms under low speed network conditions executing query Q2. The results in Figure 7.2 show that AJoin is performing much better than Hash Ripple Join in a similar way to the last experiment when Q1 is executed even though Q2 produced 528 joined tuples. On average, AJoin improves the performance by 48.4% which is slightly down from 51.8% with Q1.

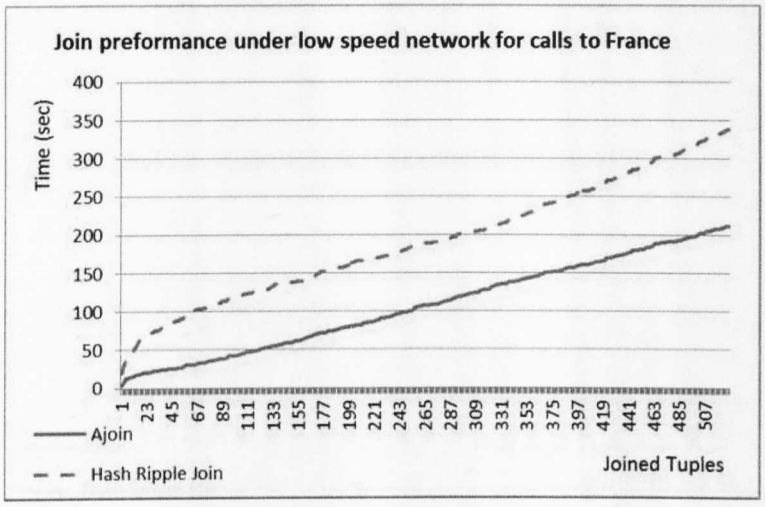

Figure 7.2 Join performance under low speed network for calls to France

The following Figure 7.3 presents the join performance of AJoin and Hash Ripple Join algorithms under low speed network conditions executing query Q3 which produces 1770 matched tuples. The results in Figure 7.3 confirm that AJoin is performing much better than Hash Ripple Join in this case too. On average, AJoin improves the performance by 49.2% which is slightly down from 51.8% with Q1, but higher than Q2. It indicates that the remote filtering could reduce the transmission cost, but it is not a only factor since that the fewer tuples are filtered ay remote site does not cause noticable changes of join performance. It is partly because the cost of transmission of join attributes is considerably lower.

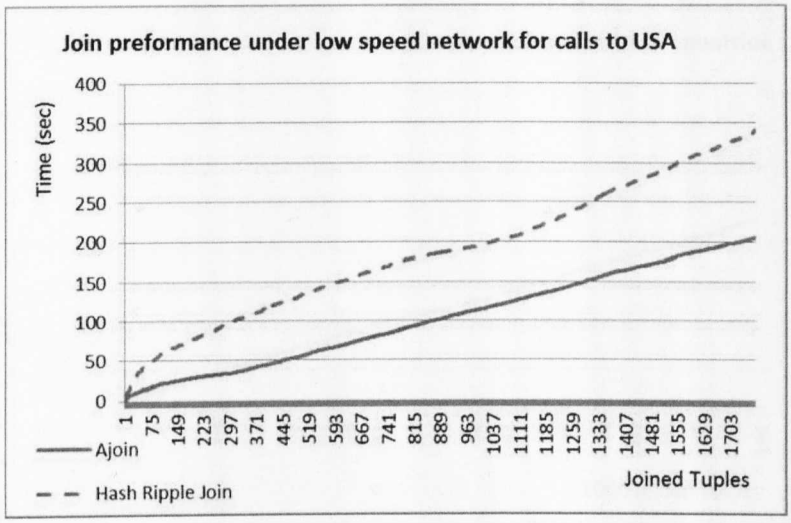

Figure 7.3 Join performance under low speed network for calls to USA

The following Figure 7.4 presents the join performance of AJoin and Hash Ripple Join algorithms under low speed network conditions executing query Q4 which produces 100,000 matched tuples. In Figure 7.4, each number on the X axis represents the number of thousands of joined tuples. The results in Figure 7.4 exhibit that AJoin is performing significantly better than Hash Ripple Join irrelevant to how many tuples will be joined. On average, AJoin improves the performance by 39.9% when all 100,000 tuples are joined.

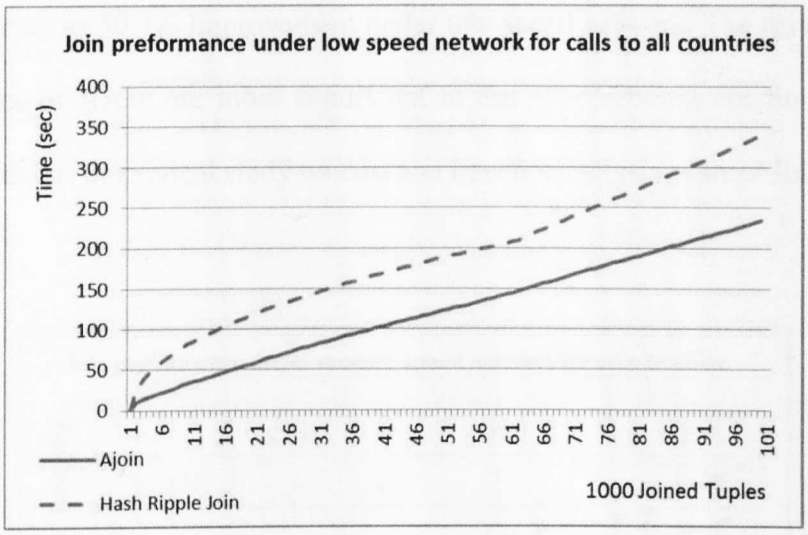

Figure 7.4 Join performance under low speed network for calls to all countries

7.2.2 AJoin Performance Under Random Network

The evaluation results shown in the following Figure 7.5 presents the join performance of AJoin and Hash Ripple Join algorithms under random network conditions. The query Q1 described in the previous section 7.2.1 is used for the evaluation.

The evaluation results shown in Figure 7.5 demonstrate that AJoin is performing remarkably better than Hash Ripple Join under random network conditions. AJoin successfully produced the first joined tuple in 4.3 seconds compared with 19.4 seconds using Hash Ripple Join. It improves the performance by 74.6%.

As the main benefits of AJoin are gained from the saving of data transmission cost during the join, comparing Figure 7.5 with Figure 7.1, the level of performance improvement is not as great as join under low speed network. On average, AJoin improves the performance remarkably by 39.8% against Hash Ripple Join, although the improvement is

not as significant as 50.7% improvement under low speed network. The results show that the advantages of AJoin are more significant in the poor network conditions, which is consistent with our theoretical study on cost and benefits analysis of the AJoin algorithm.

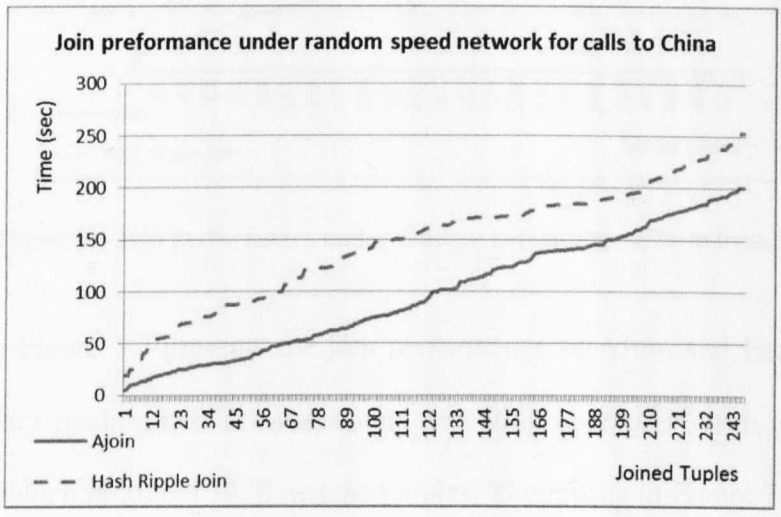

Figure 7.5 Join performance under random network for calls to China

The following Figure 7.6 presents the join performance of AJoin and Hash Ripple Join algorithms under random network conditions executing query Q2 given in the previous section 7.2.1. The results in Figure 7.6 demonstrate that AJoin is performing better than Hash Ripple Join. On average, AJoin improves the performance by 32.7% and more than 60% on the first 10 tuples.

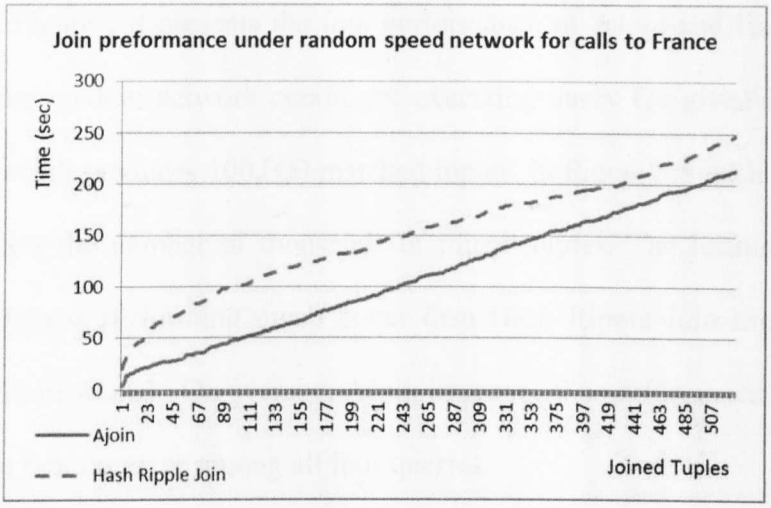

Figure 7.6 Join performance under random network for calls to France

The following Figure 7.7 presents the join performance of AJoin and Hash Ripple Join algorithms under random network conditions executing query Q3 given in the previous section 7.2.1, which produces 1770 matched tuples. The results in Figure 7.7 demonstrate that AJoin is performing much better than Hash Ripple Join. On average, AJoin improves the performance by 46.5%. Similar to the pattern of performance improvement in join under low speed network, the performance improvement of AJoin with query Q3 is between query Q1 and Q2.

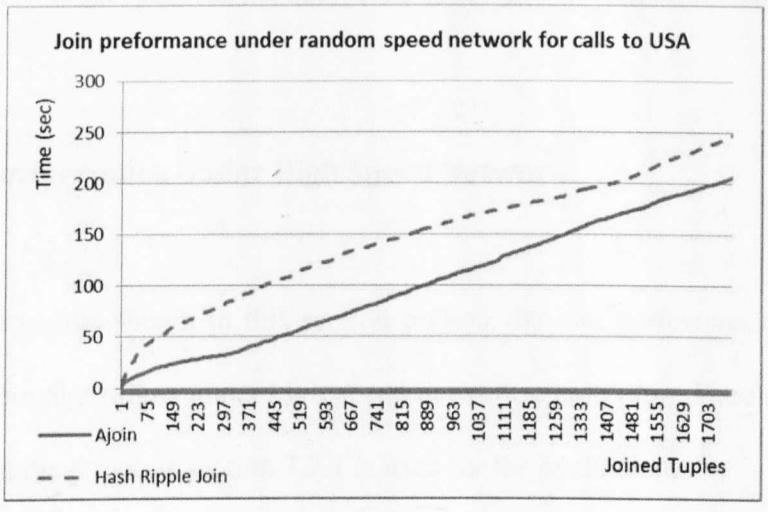

Figure 7.7 Join performance under random network for calls to USA

The following Figure 7.8 presents the join performance of AJoin and Hash Ripple Join algorithms under random network conditions executing query Q4 given in the previous section 7.2.1, which produces 100,000 matched tuples. In Figure 7.8 each number on the X axis represents the number of thousands of joined tuples. The results in Figure 7.8 exhibit that AJoin is performing much better than Hash Ripple Join irrelevant of how many tuples will be joined. On average, AJoin improves the performance by 54.3% with Q4 which is the best outcome among all four queries.

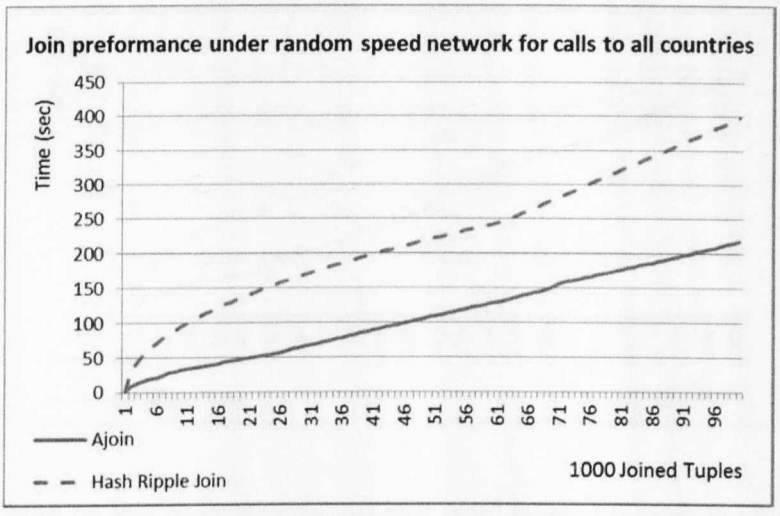

Figure 7.8 Join performance under random network for calls to all countries

7.2.3 AJoin Performance Under High Speed Network

The evaluation results shown in this section present the join performance of AJoin and Hash Ripple Join algorithms under high speed network conditions. The four queries Q1-Q4 described in the previous section 7.2.1 is used for the evaluation.

The evaluation results shown in Figure 7.9 show that AJoin is performing better than Hash Ripple Join under high network conditions. In particular, the first 10% of joined tuples are matched noticeably earlier than Hash Ripple Join. AJoin successfully produced the first joined tuple in 4.3 seconds compared with 19.4 seconds using Hash Ripple Join. Similar to join under other network models, it improves the performance significantly by 74.6%.

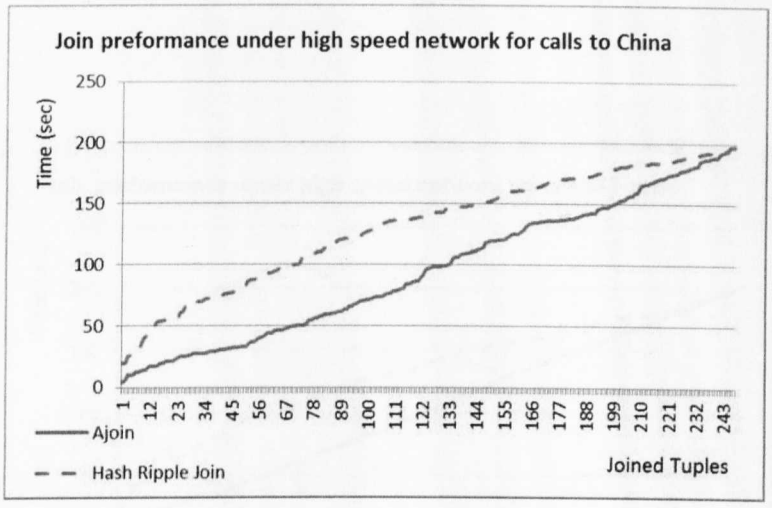

Figure 7.9 Join performance under high speed network for calls to China

Under high speed network, local data process cost will be noticeable and AJoin will automatically adapat to full join mode to avoid extra local data process cost. AJoin utilising the remote agents manages to produce the matched tuples at a much quicker rate than Hash Ripple Join at the beginning of the join stage. On average, AJoin improves the performance by 34.6% against Hash Ripple Join, although it is almost the same as Hash Ripple Join at the end of the join, since AJoin needs to wait for relation R1 to arrive, which requires almost the same amount of time as Hash Ripple Join.

The next Figure 7.10 presents the join performance of AJoin and Hash Ripple Join algorithms under high speed network conditions executing query Q2 given in the previous section 7.2.1. The results in Figure 7.10 show that AJoin is performing better than Hash Ripple Join overall. On average, AJoin improves the performance by 28.7% and it has noticeable advantages at the beginning of the join stage. For the same reason as with query Q1, AJoin finishes almost at same time as Hash Ripple Join at the end of the join.

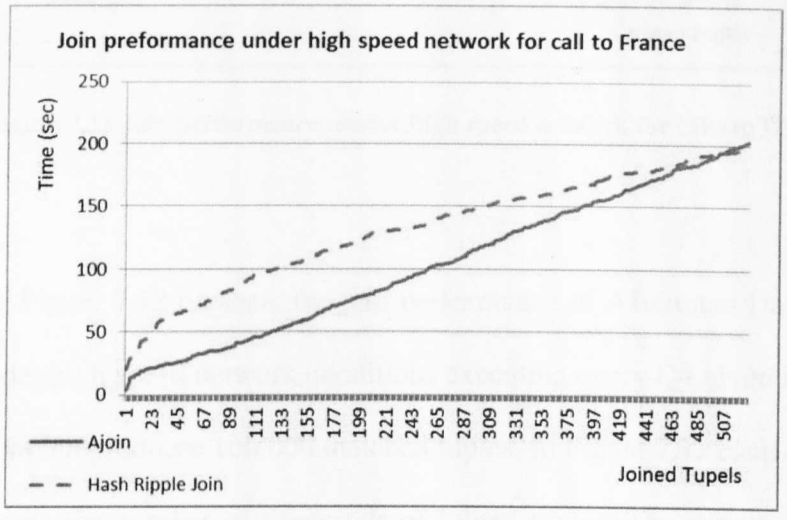

Figure 7.10 Join performance under high speed network for calls to France

The following Figure 7.11 presents the join performance of AJoin and Hash Ripple Join algorithms under high speed network conditions executing query Q3 given in the previous section 7.2.1, which produces 1770 matched tuples. The results in Figure 7.11 demonstrate that AJoin is performing better than Hash Ripple Join similar to join with query Q1 and Q2. On average, AJoin improves the performance by 41.9%. It confirms

that the slowdown of AJoin is caused by waiting of relation R1to arrive in order to complete the join process rather than the overhead of the AJoin algorithm.

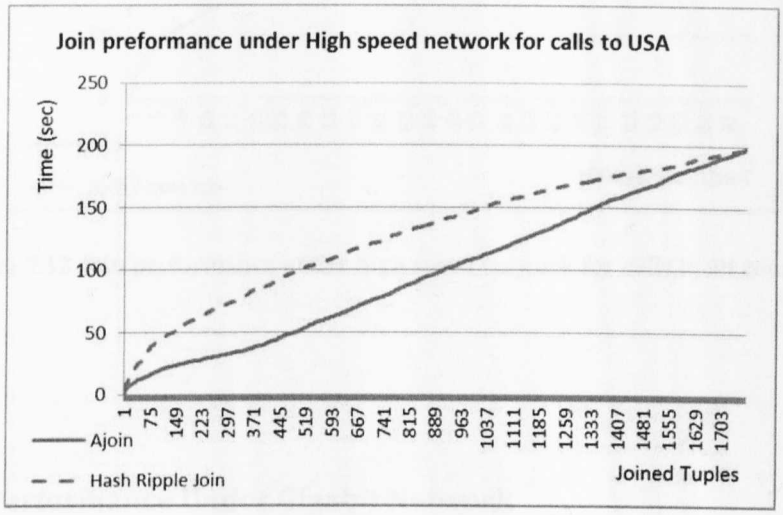

Figure 7.11 Join performances under high speed network for calls to USA

The following Figure 7.12 presents the join performance of AJoin and Hash Ripple Join algorithms under high speed network conditions executing query Q4 given in the previous section 7.2.1, which produces 100,000 matched tuples. In Figure 7.12 each number on the X axis represents the number of thousands of joined tuples. The results in Figure 7.12 exhibit that AJoin is performing the same as Hash Ripple Join. Since AJoin automatically switches to full join mode to gain the best performance and Ajoin is not able to filter extra tuples at remote site, AJoin is working almost the same as Hash Ripple Join in the case of query Q4.

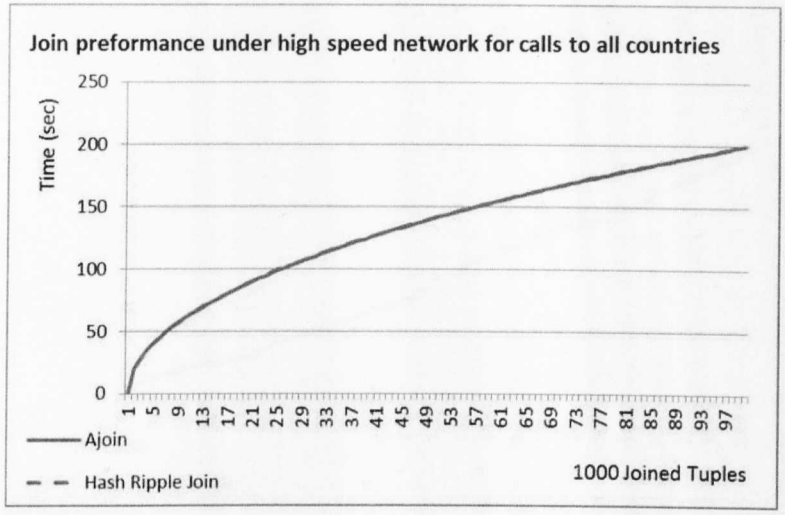

Figure 7.12 Join performance under high speed network for calls to all countries

7.2.4 AJoin Performance Under Gigabit Network

The evaluation results shown in the following Figure 7.13 - 7.16 presents the join performance of AJoin and Hash Ripple Join algorithms under Gigabit network conditions. The four queries Q1-Q4 described in the previous section 7.2.1 are used for the evaluation.

The evaluation results shown in Figure 7.13 show that AJoin is performing better than Hash Ripple Join under Gigabit network conditions. In particular, the first 10% of joined tuples are matched noticeably earlier than Hash Ripple Join. AJoin successfully produced the first joined tuple in 4.2 seconds compared with 19.3 seconds using Hash Ripple Join. Similar to join under other network models, it improves the performance significantly by 78.1%.

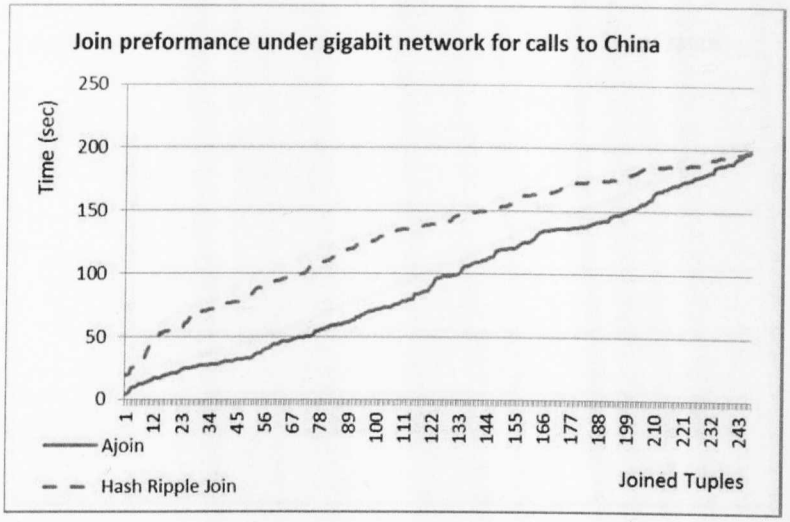

Figure 7.13 Join performance under gigabit speed network for calls to China

Comparing with Figure 7.13 with 7.9, the two figures are almost identical. This is because the transmission cost no longer has any noticeable impact on overall join performance when the data transmission speed is equivalent to local data access cost. The higher join performance of AJoin at the beginning of the join stage is mainly due to the tact that some tuples are processed at remote site by mobile agents. The same reason as AJoin under high speed network conditions, AJoin completes the join within almost the same time as Hash Ripple Join at the end of the join due to waiting for relation R1 to arrive.

As the following Figure 7.14 -7.16 are very similar to 7.10-7.12, we are not going to repeat the discussion.

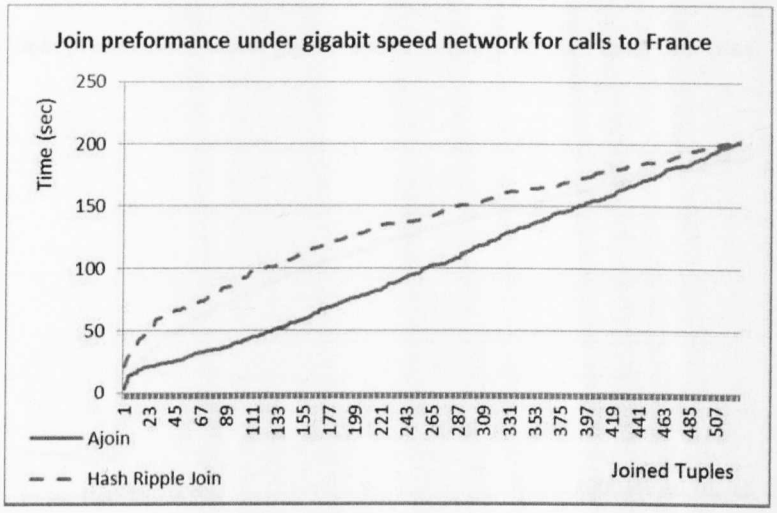

Figure 7.14 Join performance under gigabit speed network for calls to France

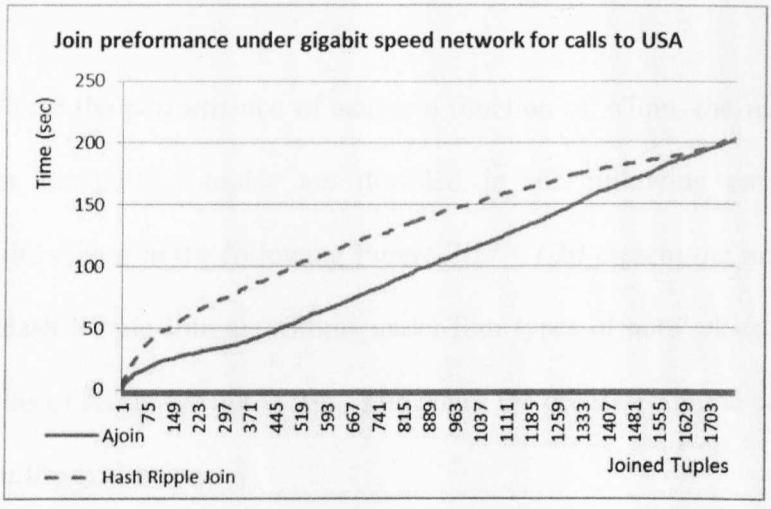

Figure 7.15 Join performance under gigabit speed network for calls to USA

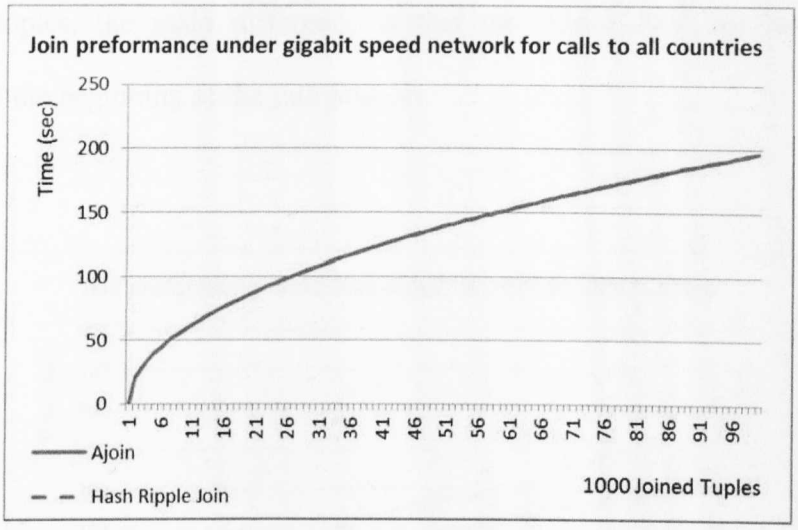

Figure 7.16 Join performance under gigabit speed network for calls to all countries

7.2.5 AJoin Performance without Remote Filtering

To further evaluate the performance of adaptive function of AJoin, the remote agents of AJoin to filter unqualified tuples are disabled in the following experiments. The evaluation results shown in the following Figure 7.17 - 7.20 present the join performance of AJoin and Hash Ripple Join algorithms under four types of network conditions where the remote agents of AJoin are not in use. The query Q1 described in the previous section 7.2.1 is used for the evaluation.

The evaluation results shown in Figure 7.17 demonstrate that AJoin is performing better than Hash Ripple Join under low speed network conditions without using the remote agents to filter unqualified tuples. The join performance has improved constantly. On average, AJoin improves the performance by 21.1%. The performance improvements are mainly gained from reducing the network transmission cost by sending only join attributes for tuple matching. Comparing the AJoin with remote agents to filter

unqualified tuples, the main difference is that the AJoin does not have significant advantages at the beginning of the join process.

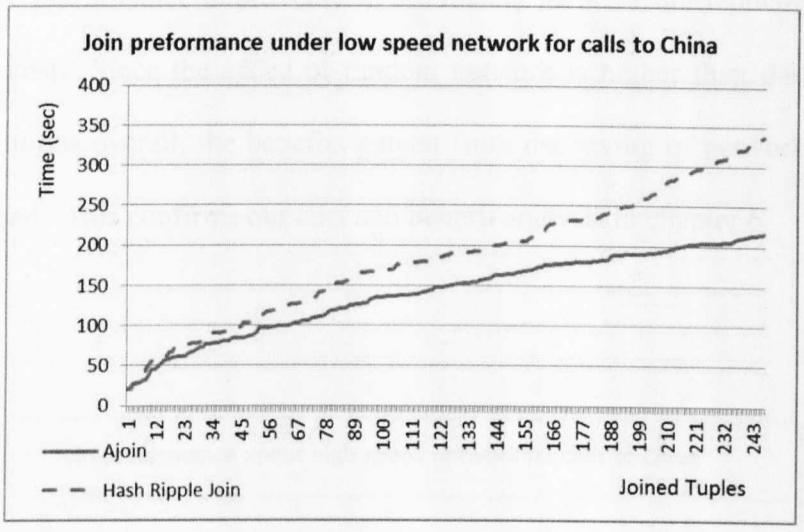

Figure 7.17 Join performance under low speed network without remote filtering

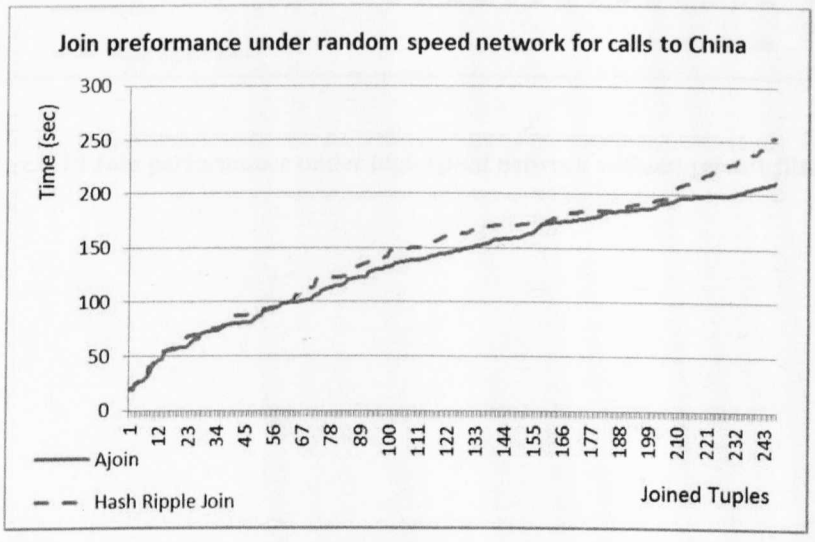

Figure 7.18 Join performances under random speed network without remote filtering

The evaluation results shown in Figure 7.18 demonstrate that AJoin is performing slightly better than Hash Ripple Join under random speed network conditions without using the remote agents to filter unqualified tuples. On average, AJoin improves the performance by 5.9%. The performance improvements are mainly gained from reducing the network transmission cost. Since the speed of random network is higher than the speed in low network conditions overall, the benefits gained from the saving of network transmission cost are reduced. This confirms our cost and benefit analysis in chapter 6.

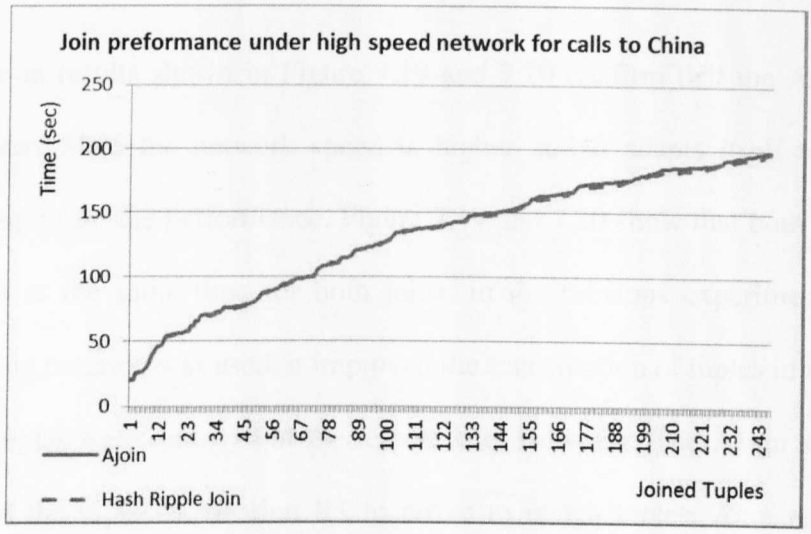

Figure 7.19 Join performance under high speed network without remote filtering

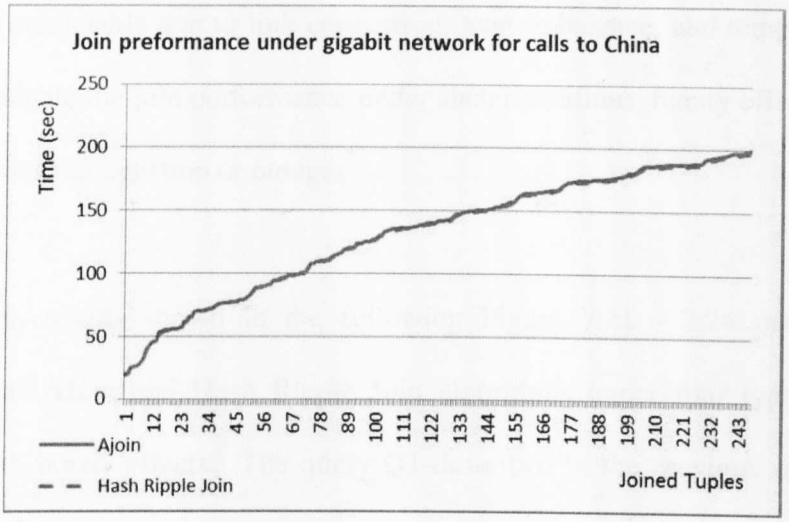

Figure 7.20 Join performance under gigabit network without remote filtering

Both evaluation results shown in Figure 7.19 and 7.20 confirm that the AJoin does not cause overhead when the network speed is higher and it adapts itself to a best join approach to optimise the performance. Figure 7.19 and 7.20 show that both relations will arrive almost at the same time for both joins. In the previous experiments, when the remote filtering function was used, it improved the transmission of tuples in relation R2 as unqualified tuples were removed at the remote site. However, The AJoin algorithm still needs to wait the tuples in relation R1 to arrive to match tuples. As a result, the join completion time for both AJoin and Hash Ripple join are almost the same. It confirms that in Figure 7.9 and 7.13, the slowdown of AJoin is simply caused by waiting for relation R1 to arrive in order to complete the join process rather than the overhead of the AJoin algorithm. The situation will not deteriorate even if more tuples need to be joined.

7.2.6 AJoin Performance with Bursty Effects

In wide-area distributed network environments such as the Internet where data access

becomes less predictable due to link congestion, load imbalance, and temporary outages. In order to evaluate the join performance under such conditions, bursty effects are used to model the network congestion or outages.

The evaluation results shown in the following Figure 7.21 - 7.24 present the join performance of AJoin and Hash Ripple Join algorithms under four types of network conditions with bursty effects. The query Q1 described in the previous section 7.2.1 is used for the evaluation.

The evaluation results shown in the following Figure 7.21 demonstrate that AJoin is performing significantly better than Hash Ripple Join under slow speed network with bursty effects. Compared with Hash Ripple Join, AJoin improves the performance by 57.0% on average. The performance improvements are mainly gained from reducing the network transmission cost. The overall time used for join is longer than slow speed network due to the delay caused by burst, but both Figure 7.21 and 7.1 show a similar performance improvement pattern.

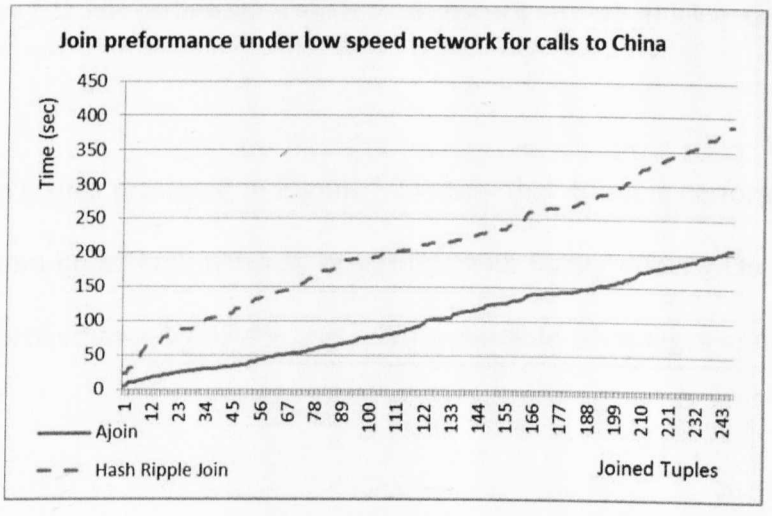

Figure 7.21 Join performance under low speed network with bursty effects

The evaluation results shown in the following Figure 7.22 demonstrate that AJoin is performing noticeably better than Hash Ripple Join under random speed network with bursty effects. Compared with Hash Ripple Join, AJoin improves the performance by 49.2% on average with similar a pattern of join under random speed network. The performance improvement is up from 32.7 comparing with join under random speed network without bursty effects since more savings are generated when the network is slowed down due to burst.

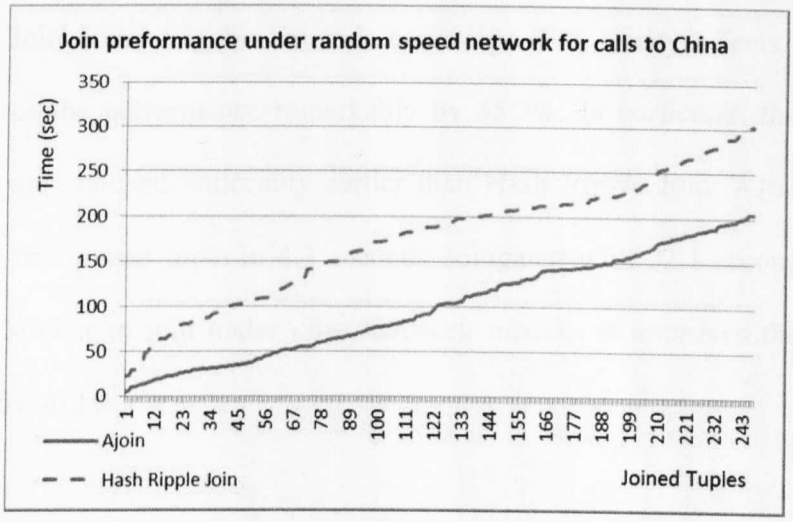

Figure 7.22 Join performances under random speed network with bursty effects

The evaluation results presented in Figure 7.23 show that AJoin is performing better than Hash Ripple Join under high network conditions with bursty effects. On average, AJoin improves the performance by 43.4% and it has noticeable advantages in the beginning of the join stage.

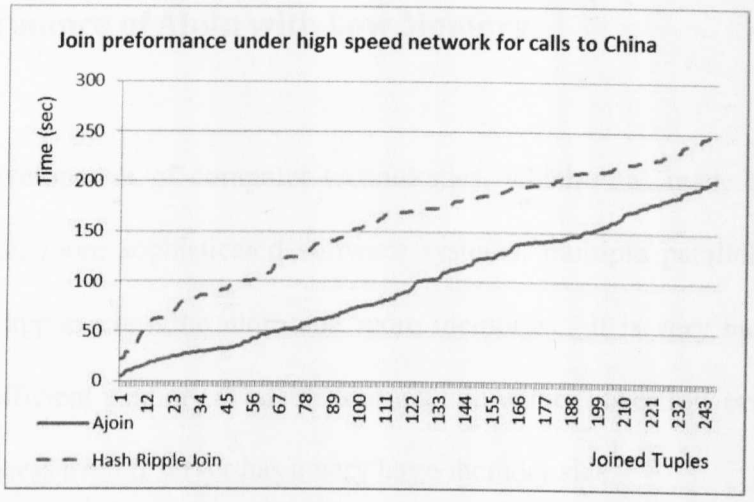

Figure 7.23 Join performances under high speed network with bursty effects

The evaluation results presented in Figure 7.24 show that AJoin is performing better than Hash Ripple Join under Gigabit network conditions with bursty effects. On average, AJoin improves the performance remarkably by 45.7%. In particular, the first 10% of joined tuples are matched noticeably earlier than Hash Ripple Join. AJoin successfully produced the first joined tuple in 4.3 seconds compared with 22.1 seconds using Hash Ripple Join. Similar to join under other network models, it improves the performance significantly by 80.1%.

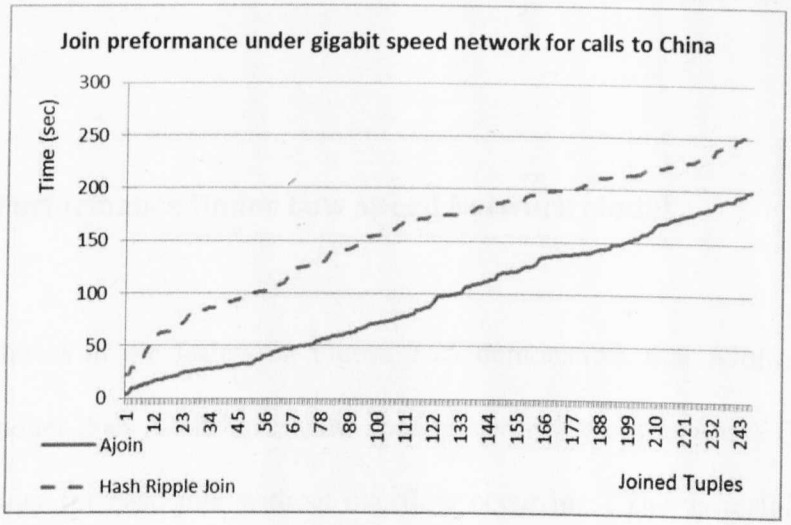

Figure 7.24 Join performances under gigabit speed network with bursty effects

7.3. Performance of AJoin with Low Memory

Despite the development of computer technologies, which has made more memories available to use, more sophisticated software systems, multiple parallel processes and multiple user support consume more and more memories. It is very hard to guarantee there will be sufficient memory available to run a join without overflow occurring in Hash Ripple Join process even if server has a very large memory size.

To ensure the join process could still run effectively when the available memory space becomes lower, XJoin was proposed to address the issue. According to our previous study, XJoin is one of the best join algorithms able to run in various memory conditions.

The evaluation results presented in the following sections are the outcomes of the evaluation of effectiveness and performance of AJoin algorithm in lower memory conditions. Similar to the evaluation reported in section 7.2, AJoin will be evaluated against XJoin under four simulated network environments with four different sizes of memory. The query Q1 described in section 7.2 will be used for evaluation.

7.3.1 AJoin Performance Under Low Speed Network Model

The results shown in the following Figure 7.25 demonstrate that AJoin is performing significantly better than XJoin under low speed network executing query Q1 with 5% of required memory for hash join without overflow occurring. This is mainly because the hash table used by AJoin only contains join attributes which is much smaller than all

attributes required by XJoin. With the same memory space AJoin can process as much as 10 to 40 times more tuples. Although AJoin requires secondary storage to store complete attributes for a tuple, it does not cause significant overhead as most of the attributes in a tuple will only be required when the tuple is matched and required to output the join results.

AJoin also gains benefits from data transmission over low speed network as well as remote data processing as we discussed in 7.2. Compared with XJoin, AJoin improves the performance on average by 71.5% with 5% of memory space available.

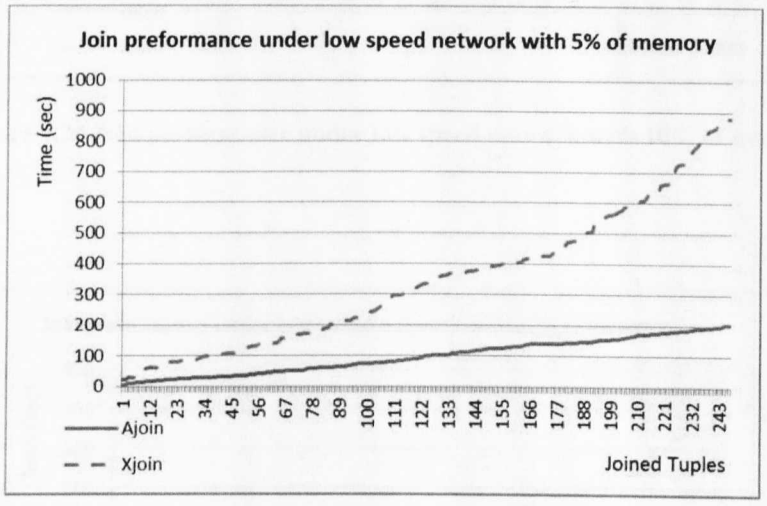

Figure 7.25 Join performance under low speed network with 5% of memory

The evaluation results shown in the following Figure 7.26 - 7.28 demonstrate that AJoin is performing significantly better than XJoin under low speed network with 10%, 20%, and 50% of memory space available. Compared with Figure 7.25, AJoin on one hand almost performs the same as previously as AJoin already has enough memory to run the join process smoothly, while XJoin on the another hand has improved its performance

significantly due to the larger memory space. However, compared with XJoin, AJoin still improves the performance on average by 53.4%, 54.3%, and 53.0% with 10%, 20% and 50% of memory space available respectively. The main reasons of the advantages of AJoin are the same as we discussed in the previous section.

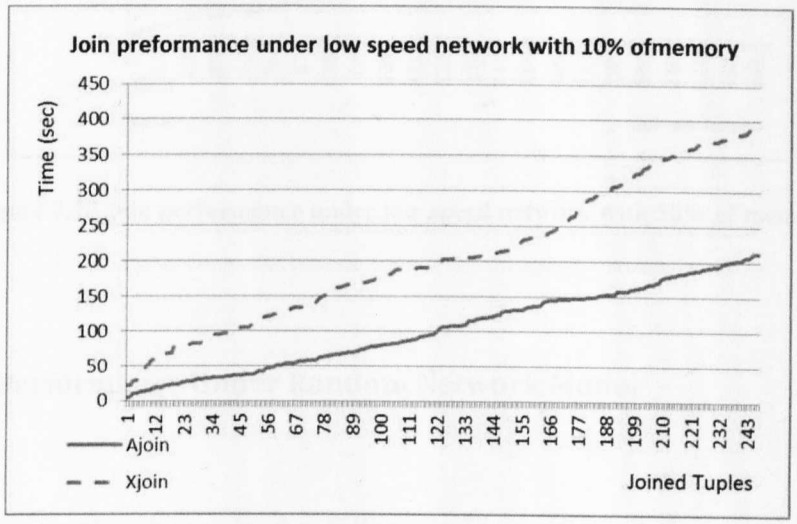

Figure 7.26 Join performance under low speed network with 10% of memory

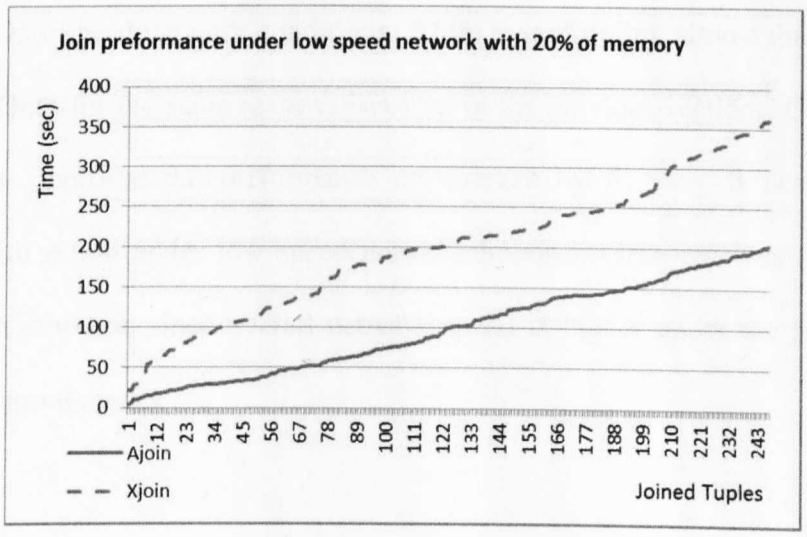

Figure 7.27Join performance under low speed network with 20% of memory

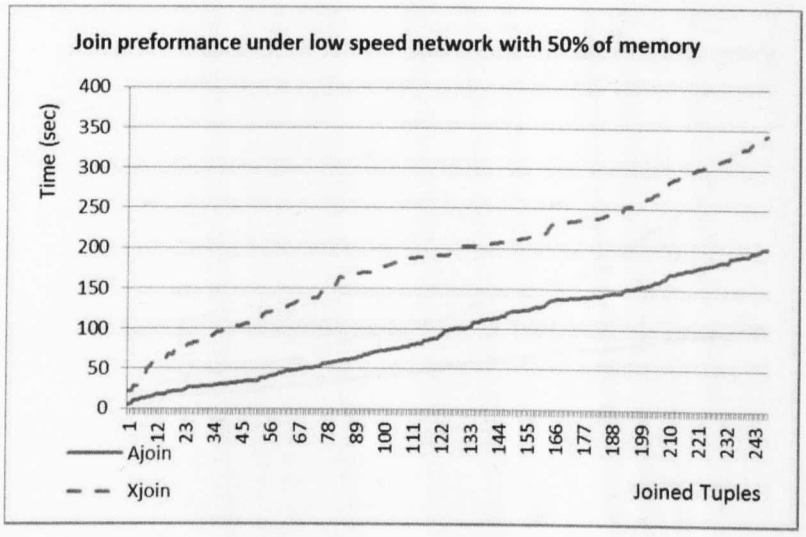

Figure 7.28 Join performance under low speed network with 50% of memory

7.3.2 AJoin Performance Under Random Network Model

The evaluation results shown in the following Figure 7.29 demonstrate that AJoin is performing significantly better than XJoin under random speed network executing query Q1 with 5% of required memory for hash join without overflow occurring. Similar to AJoin under low speed network conditions, AJoin is performing almost the same and far better than XJoin for the same reason discussed in the previous section. Compared with XJoin, AJoin improves the performance on average by 61.8%. It is slightly down compared with AJoin under low speed network because AJoin receives fewer benefits from data transmission since overall network speed is higher under the random model than the low speed model.

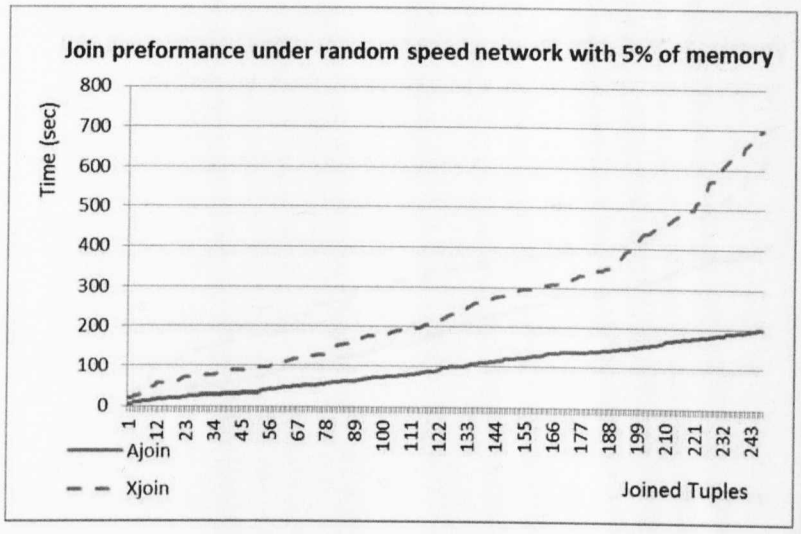

Figure 7.29 Join performance under random speed network with 5% of memory

The evaluation results shown in the following Figure 7.30 demonstrate that AJoin is performing noticeably better than XJoin under low speed network with 10% of memory space available. Compared with Figure 7.29, AJoin almost performs the same as previously since AJoin already has enough memory to run the join process smoothly. XJoin has improved its performance significantly due to the larger memory space. However, XJoin slows down noticeably in its third join stage as some tuples have to join tuples from secondary storage. Compared with XJoin, AJoin improves the performance on average by 42.3% with 10% of memory space available.

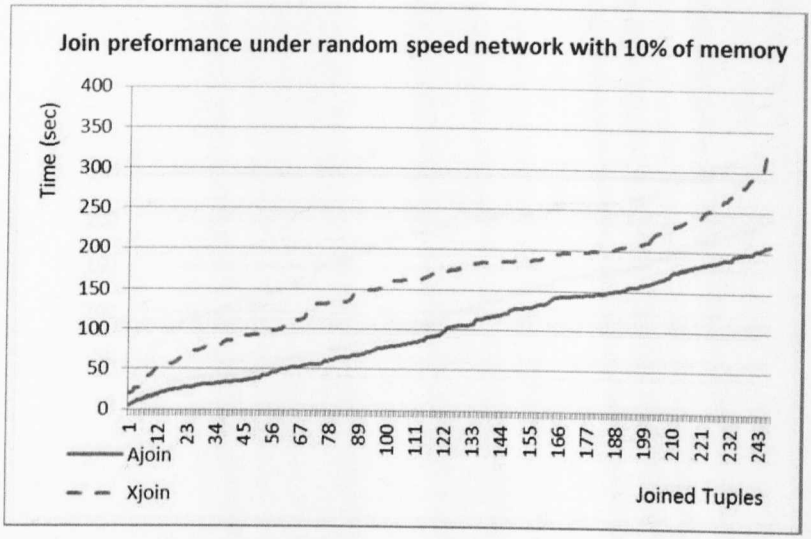

Figure 7.30 Join performance under random speed network with 10% of memory

The evaluation results shown in the following Figure 7.31 - 7.32 demonstrate that AJoin is performing noticeably better than XJoin under low speed network with 20%, and 50% of memory space available. Compared with Figure 7.30, XJoin has further improvements in its performance due to the larger memory space and its performance is very close to Hash Ripple Join. Comparing with XJoin, AJoin improves the performance on average by 50.4%, and 43.4% with 20% and 50% memory space available respectively. The main reasons of the advantages of AJoin are the same as we discussed in the previous session.

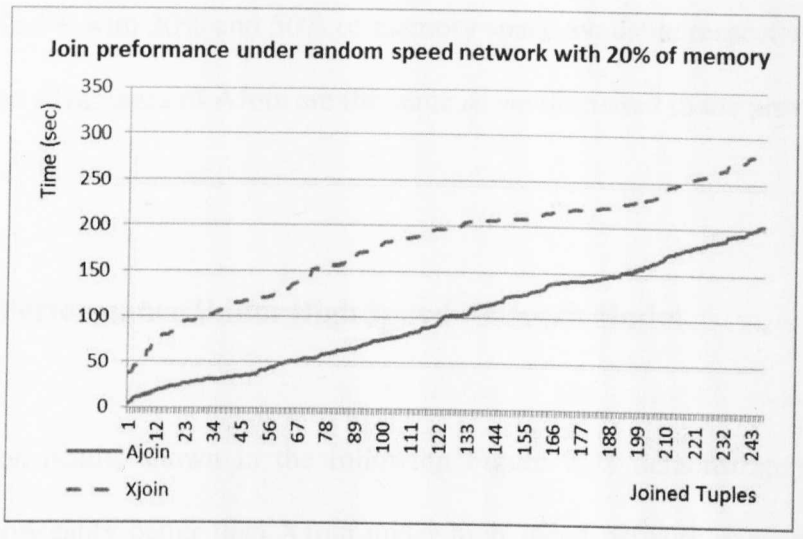

Figure 7.31 Join performance under random speed network with 20% of memory

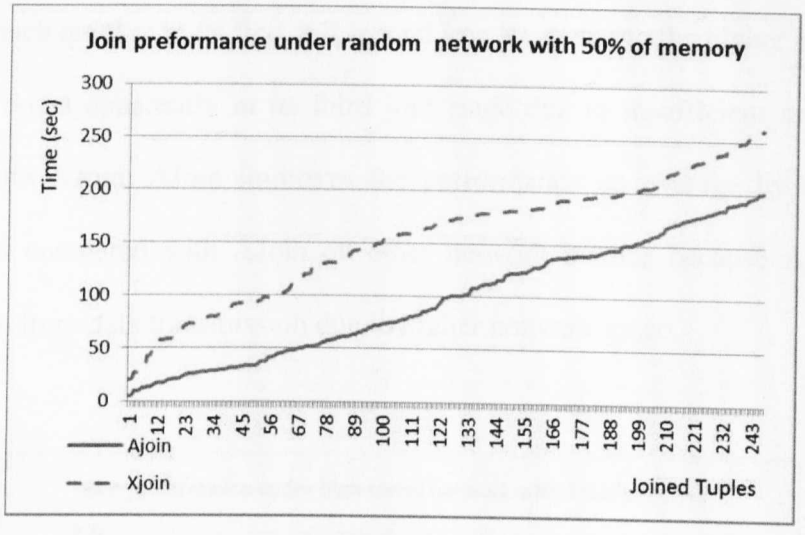

Figure 7.32 Join performance under random speed network with 50% of memory

The evaluation results shown in the following Figure 7.31 - 7.32 demonstrate that AJoin is performing noticeably better than XJoin under low speed network with 20%, and 50% of memory space available. Compared with Figure 7.30, XJoin has further improvements in its performance due to the larger memory space and its performance is very close to Hash Ripple Join. Compared with XJoin, AJoin improves the performance on average by

50.4%, and 43.4% with 20% and 50% of memory space available respectively. The main reasons for the advantages of AJoin are the same as we discussed in the previous section.

7.3.3 AJoin Performance Under High Speed Network Model

The evaluation results shown in the following Figure 7.33 demonstrate that AJoin is performing noticeably better than XJoin under high speed network executing query Q1 with 5% of required memory for hash join without overflow occurring. Similar to AJoin under low speed network conditions, AJoin is performing almost in an identical pattern. XJoin runs much quicker in its first and second join stage due to the higher network speed, it is slowed down apparently in its third join stage due to insufficient memory space. Compared with XJoin, AJoin improves the performance on average by 49.3%. It is slightly down compared with AJoin on other network models because AJoin receives fewer benefits from data transmission due to higher network speed.

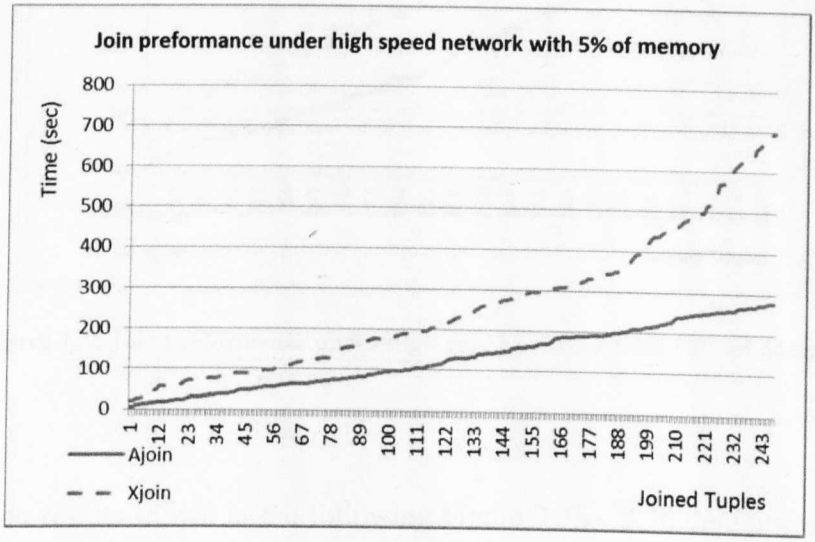

Figure 7.33 Join performance under high speed network with 5% of memory

The evaluation results shown in the following Figure 7.34 demonstrate that AJoin is performing noticeably better than XJoin under low speed network with 10% of memory space available. AJoin takes advantages of remote process to filter unqualified tuples before the join process, which has made significant improvements in the beginning of the join process to produce joined tuples much earlier. Compared with Figure 7.33, AJoin almost performs the same, while XJoin has improved its performance significantly due to the larger memory space. However, XJoin slows down noticeably in its third join stage where remaining tuples have to join from secondary storage. Compared with XJoin, AJoin improves the performance on average by 31.9% with 10% of memory space available.

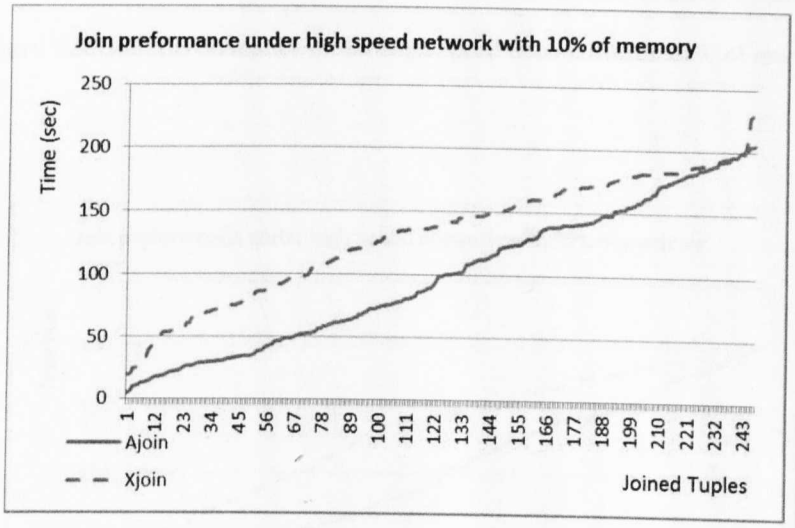

Figure 7.34 Join performance under high speed network with 10% of memory

The evaluation results shown in the following Figure 7.35 - 7.36 demonstrate that AJoin is performing noticeably better than XJoin under low speed network with 20%, and 50% memory space available. Compared with Figure 7.34, XJoin has further improved its

performance due to the larger memory space and its performance is very close to Hash Ripple Join. Compared with XJoin, AJoin improves the performance on average by 33.6% and 31.3% with 20% and 50% of memory space available respectively. The main reasons for the advantages of AJoin are the same as we discussed in the previous section.

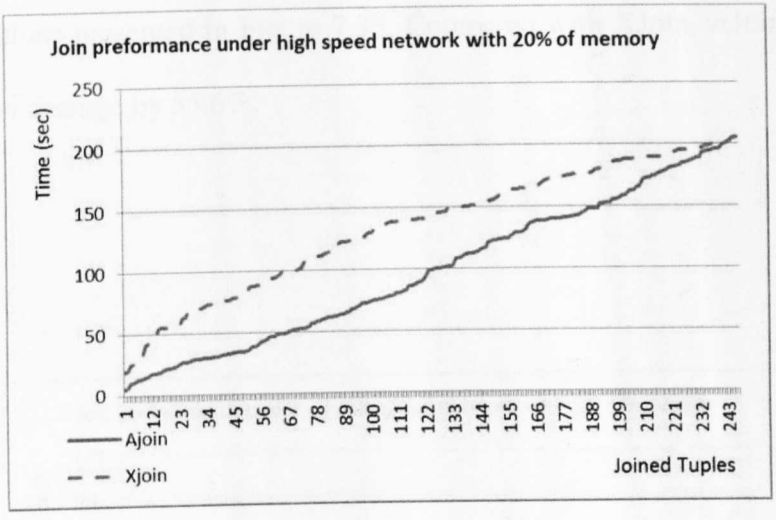

Figure 7.35 Join performance under high speed network with 20% of memory

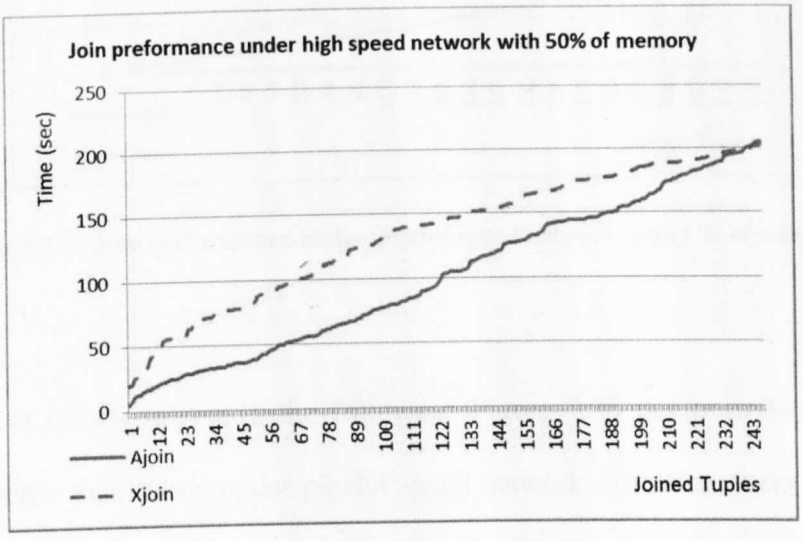

Figure 7.36 Join performance under high speed network with 50% of memory

7.3.4 AJoin Performance Under Gigabit Speed Network Model

The evaluation results shown in the following Figure 7.37 demonstrate that AJoin is performing noticeably better than XJoin under gigabit speed network executing query Q1 with 5% of required memory for hash join without overflow occurring. The pattern of join process for both AJoin and XJoin are very similar to the join under high speed network conditions presented in Figure 7.33. Compared with XJoin, AJoin improves the performance on average by 55.6%.

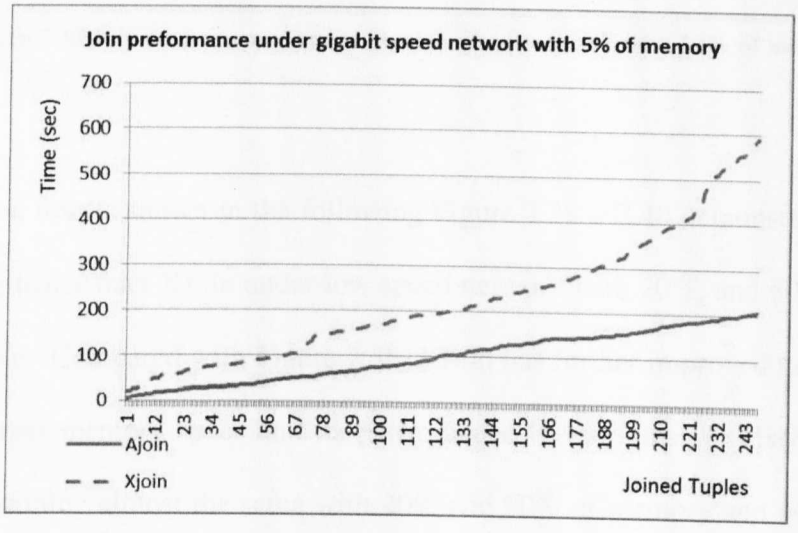

Figure 7.37 Join performance under gigabit speed network with 5% of memory

The evaluation results shown in the following Figure 7.38 demonstrate that AJoin is performing better than XJoin under gigabit speed network executing query Q1 with 10% of required memory for hash join without overflow occurring. The pattern of join process for both AJoin and XJoin are very similar to the join under high speed network conditions

presented in Figure 7.34. Compared with XJoin, AJoin improves the performance on average by 29.1%.

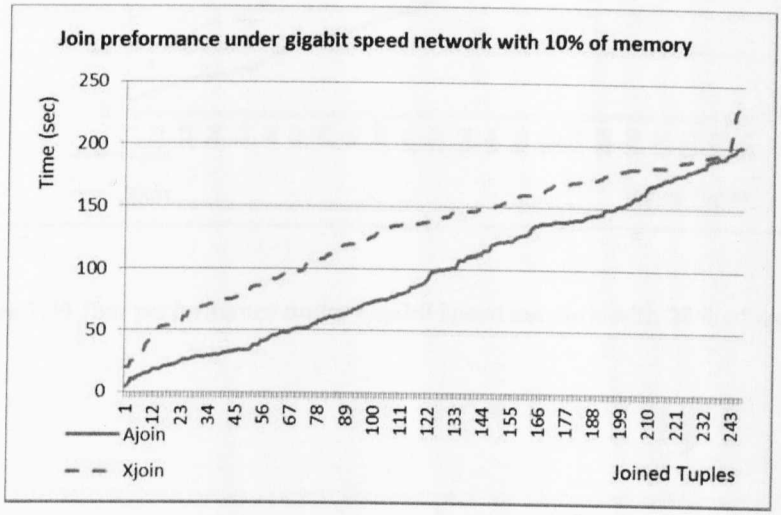

Figure 7.38 Join performance under gigabit speed network with 10% of memory

The evaluation results shown in the following Figure 7.39 – 7.40 demonstrate that AJoin is performing better than XJoin under low speed network with 20%, and 50% of memory space available. Compared with Figure 7.38, XJoin has further improved its performance due to the larger memory space and its performance is very close to Hash Ripple Join. AJoin is performing almost the same with 20% and 50% of memory and no extra saving gained from data transmission in both memory conditions. Compared with XJoin, AJoin improves the performance on average by 31.2% and 33.2% with 20% and 50% of memory space available respectively. The advantages are largely gained from remote process.

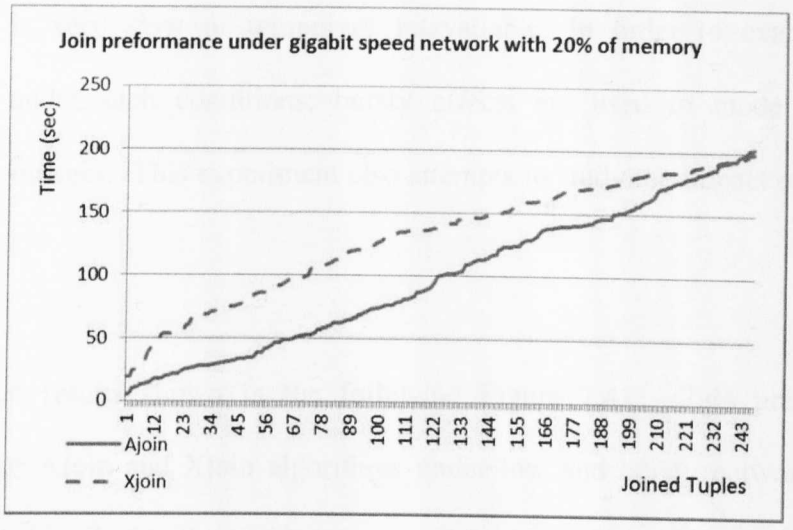

Figure 7.39 Join performance under gigabit speed network with 20% of memory

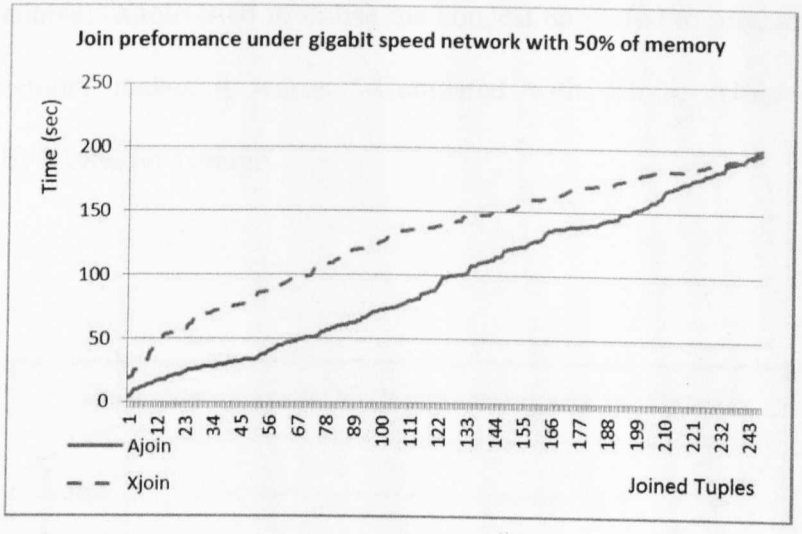

Figure 7.40 Join performance under gigabit speed network with 50% of memory

7.3.5 AJoin Performance with Bursty Effects

Xjoin has an especially designed second join stage to tackle wide-area distributed network issues related to congestion or temporarily outages. XJoin could utilise the period when

the network is very slow or temporary unavailable. In order to evaluate the join performance under such conditions, bursty effects are used to model the network congestion or outages. This experiment also attempts to study the impact on the memory availability.

The evaluation results shown in the following Figure 7.41 - 7.44 present the join performance of AJoin and Xjoin algorithms under low and bursty network conditions. The queriesQ1 described in the previous section 7.2.1 is used for the evaluation.

The evaluation results shown in the following Figure 7.41 demonstrate that AJoin is performing significantly better than XJoin under slow bursty network with 5% of memories available. XJoin tried to utilise the congestion period to produce extra tuples, but lower memory makes it worse. Compared with XJoin, AJoin improves the performance by 73.8% on average.

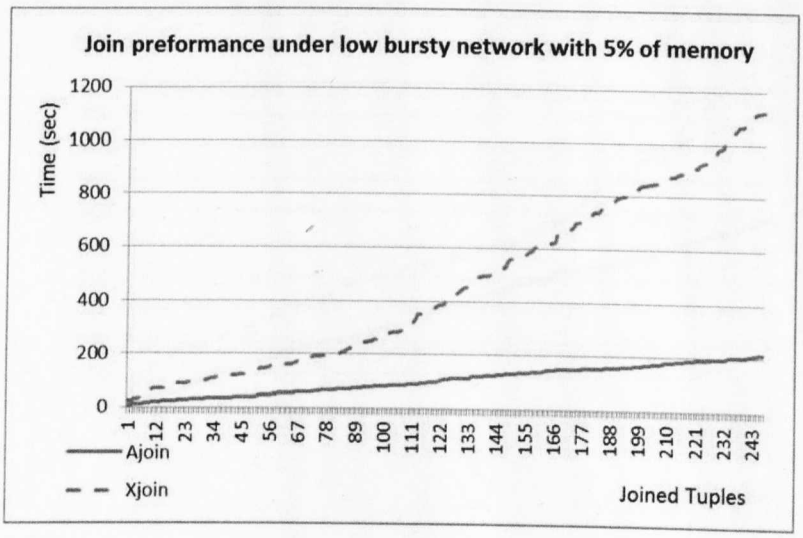

Figure 7.41 Join performance under low bursty network with 5% of memory

This is mainly because the AJoin needs a much smaller memory than XJoin. AJoin also gains benefits from data transmission over low speed network as well as remote data processing as we discussed in 7.2. The modern slow network is not the same as the traditional 'slow' network when XJoin was designed. The network speed dropping to 1kb/s may not be sufficient to provide enough time for XJoin to benefit during its second join operation.

The evaluation results shown in the following Figure 7.42 - 7.44 demonstrate that AJoin is performing significantly better than XJoin under slow bursty network with 10%, 20%, and 50% of memory space available. Compared with Figure 7.41, XJoin has noticeable improvement of its performance due to the larger memory space. Performance of XJoin with 10% of memory shown on Figure 7.72 indicates that it has a significant improvement in its first and second join stages, but it is suffering on the third stage because of lower memories. Compared with XJoin, AJoin improves the performance by 41.2% on average.

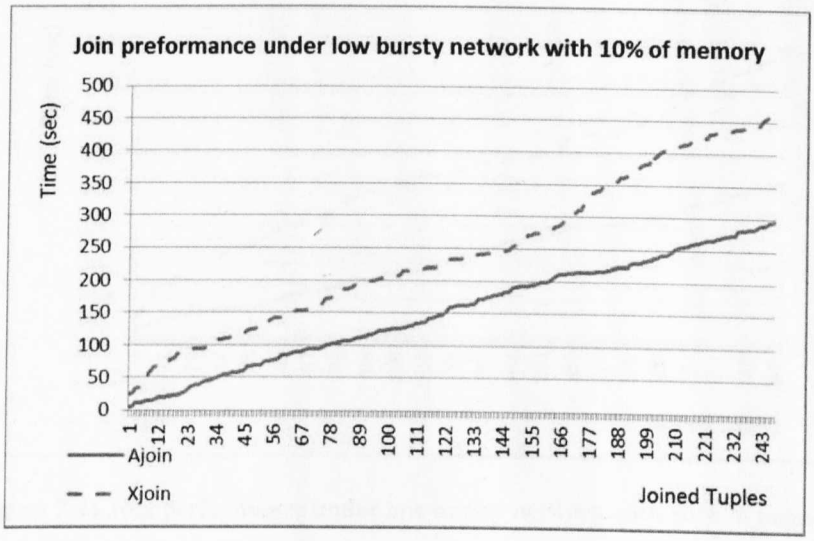

Figure 7.42 Join performance under low bursty network with 10% of memory

When the memory is increased to 20% and 50%, Xjoin is performing smoothly. However, compared with AJoin, the performance of XJoin is behind on average by 49.8% and 55.0% with 20% and 50% of memory space available respectively. The main reasons for the advantages of AJoin are the same as we discussed in the previous section.

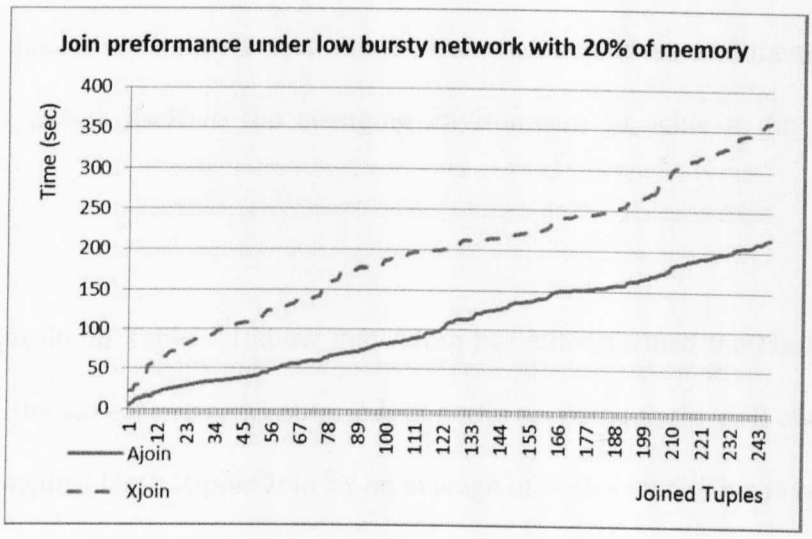

Figure 7.43 Join performance under low bursty network with 20% of memory

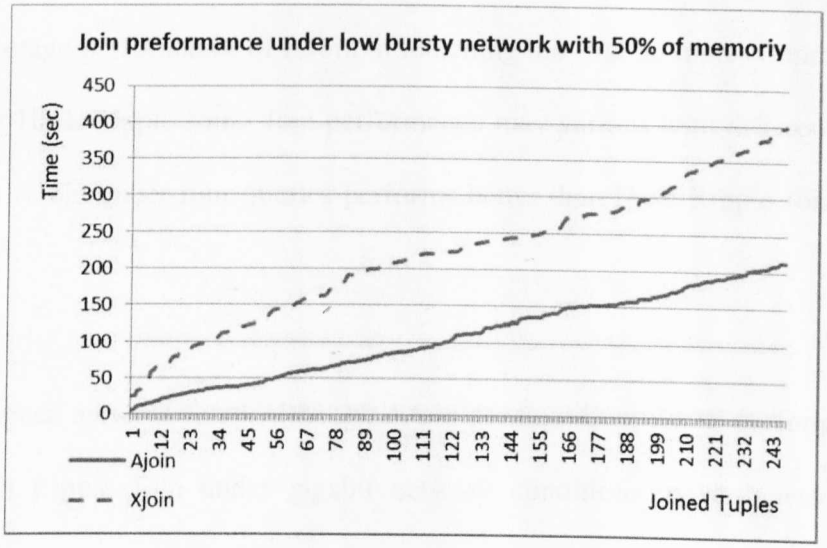

Figure 7.44 Join performance under low bursty network with 50% of memory

7.4. Summary

The evaluation results discussed in the previous sections can be summarised into the table 7.1 and 7.2. The results demonstrate that AJoin has significant scalability and adaptivity. It performs consistently outperformed in all of network conditions and memory spaces. It automatically adapts itself to the changing environment to achieve the best possible performance.

The overall results in Table 7.1 show that AJoin has outperformed the Hash Ripple Join algorithm. In the slower network setup, AJoin performs particularly well and it improves performance against Hash Ripple Join by an average of 44% - 49%. This is because AJoin dynamically adapts to the network speed by selecting appropriate join methods and utilising remote agents to filter unqualified tuples in the remote site in order to reduce data transmission cost. In addition, AJoin produces efficient matching at an earlier stage of the join. The average performance of AJoin in matching the first 50 tuples improves as much as 67% over Hash Ripple Join. Join performance may various with different queries. On average, the AJoin under four queries performs better than Hash Ripple Join by 23.6% - 48.4 %.

In the high speed network setup, although AJoin does not improve its performance further against Hash Ripple Join under gigabit network conditions, nevertheless it does not perform any worse. on average, AJoin also performs better than the Hash Ripple Join by 29.7 - 30.3%, as AJoin is facilitated by parallel processing at the local and remote sites.

AJoin Performance Improvements Against Hash Ripple Join

Network / Other conditions	Low Speed(%)	Random(%)	High Speed(%)	Gigabit(%)	Average(%)
Q1 - Calls to China	51.8	39.8	34.6	34.9	40.275
Q2 - Calls to France	48.4	32.7	28.7	31.1	35.225
Q3 - Calls to USA	49.2	46.6	41.9	39.7	44.35
Q4 - Calls to all countries	39.9	54.3	0.22	0	23.605
Q1 With Bursty Effects	56.4	48.2	43.4	45.7	48.425
Average	49.14	44.32	29.764	30.28	38.376

Table 7.1 AJoin performance improvements against Hash Ripple Join

The overall results in Table 7.2 show that AJoin performance improvement against XJoin under various network environments and memory available spaces. On average, the AJoin under four network models performs remarkably better than XJoin. AJoin improves performance by 36.5% – 57.8%. AJoin improves performance more under lower and unpredictable network conditions. This is because the main benefits of AJoin are gained from the saving of data transmission cost during the join. The slower network generates the more benefits. AJoin performance may various with memory available spaces. On average, the AJoin under four memory available spaces outperforms XJoin by 40.4% - 62.2 %. This is because AJoin requires much less memory to operate the join process that XJoin.

AJoin Performance Improvements Against XJoin

Network / Memory	Low Speed(%)	Random(%)	High Speed(%)	Gigabit(%)	Low Speed with busrt(%)	Average(%)
5% of Memory	70.5	61.83	49.3	55.6	73.8	62.206
10% of Memory	53.4	42.3	31.9	33.2	41.2	40.4
20% of Memory	54.3	50.4	33.6	31.9	49.8	44
50% of Memory	53	43.4	31.3	33.2	55	43.18
Average	57.8	49.4825	36.525	38.475	54.95	47.4465

Table 7.2 AJoin performance improvements against XJoin

It should be noted that the benefits of AJoin are linked to the network speed. The faster the network speed, the less tangible benefits AJoin could generate. However, the speed is measured related to local process speed rather than an absolute bandwidth value. With the availability of more powerful CPUs and faster access to secondary storage devices, AJoin will be able to obtain benefits from higher network speed due to higher local process speed.

The real network speeds are far slower than their theoretical bandwidth values due to network latency, protocol overhead, driver efficiency, and a range of other technical issues. In general, real transfer speeds are expected to be 40 to 80% of the maximum speed. The real file transmission speed using VPN with the Internet connections is much lower. According to (vpnsp.com, 2011) VPN Speed Test Results, the top 10 service providers can only deliver 26 – 68 % of their baseline speed with maximum speed 27.87 Mb/s. Therefore, AJoin could gain more benefits in real network environments.

Chapter 8. Conclusions and Further Research

8.1. Summary

In this thesis, we have discussed the background of data warehouses and investigated the evolution of data warehouses to identify architecture suitable for highly distributed data warehouses. The feasibility and effectiveness of utilising software agent technology for distributed information systems have been studied. Based on the research and investigation, we have proposed an agent-based data warehouse architecture which use software agent to integrate a dynamic integration approach with traditional data warehousing technologies seamlessly to address the issues arising from distributed and dynamic data warehouse environments. In the agent-based data warehouse, data warehousing functions are organised in a multi-agent platform where software agents are working intelligently and autonomously to adapt to a dynamic environment and cooperating each other to provide better performance. Join operation has been identified as an important component in making the architecture successful. In order to obtain better join performance under distributed and dynamic data warehouse environments, and to provide users promptly with query results, we proposed an agent-based adaptive join algorithm called AJoin for effective and efficient online join operations in distributed data warehouses. Taking into consideration data warehouse features, AJoin utilises intelligent agents for dynamic optimisation and coordination of join processing at run time. Key aspects of the AJoin algorithm have been implemented and evaluated against other modern adaptive join algorithms. It has been shown that AJoin exhibits significantly better performance under various distributed and dynamic data warehouse environments in our study.

The main work undertaken in this PhD research is summarised below:

- Literature review on both data warehousing and software agents

  To investigate the evolution of data warehouses architecture in order to identify architecture suitable for highly distributed data warehouses and the feasibility and effectiveness of utilising software agent technology to address some specific issues in data warehouses.

- Proposed framework for agent-based data warehouse architecture

  Agent-based data warehouse architecture was proposed to tackle the real-time integration problem in distributed and dynamic data warehouse environments. It is believed that the problem arising from distributed and dynamic data warehouse environments can be better tackled in the agent-based data warehouse architecture. Join operation as a key operation for the architecture has been identified.

- Pilot study of join algorithms for data warehouse

  Modern adaptive join approaches as one of the most important essential techniques in distributed and dynamic data warehouse environments are focused on and investigated. A pilot study has been carried out to seek the most effective and efficient online join algorithms. Four modern adaptive join algorithms are implemented and experimented in four types of simulated network environments with various join conditions.

- Establishment and construction of data warehouse environment

  A typical data warehouse environment based on a real industrial case has been simulated for the study and evaluation of data warehouse processing and various join approaches effectively.

- Proposed AJoin, an adaptive join algorithm using intelligent agents

  The findings from the pilot study have indicated that modern join algorithms have very good adaptability to tackle the issues arisen from unpredictable network environment, but unfortunately they are designed for generic data join purpose and have been optimised for data warehousing. Therefore, an adaptive pipelined join algorithm called AJoin was proposed to provide effective and efficient online join algorithm for distributed data warehouses in dynamic environments. The main algorithm of AJoin was implemented and evaluated against other modern join algorithms. AJoin has exhibited better performance under distributed and dynamic data warehouse environments. The outcome of this research has been very encouraging and the findings were presented at Computation World, November 2009, Athens, Greece and published at the IEEE Digital Library (Qicheng Yu et al., 2009).

The work undertaken for this PhD research has provided a solid basis for the development of an agent-based approach for data warehousing which addresses issues arising from current distributed and dynamic data environments. A general framework for agent-based data warehouse architecture has been presented. Adaptive join algorithms as a key technique in this new environment have been investigated and their performance evaluated. In particular, a new agent-based join algorithm, AJoin, has been proposed to

improve query processing performance in the dynamic and distributed data warehouse context. The main algorithm of AJoin was implemented and evaluated against other modern join algorithms. AJoin has exhibited better performance under distributed and dynamic data warehouse environments. The outcome of this research has been very encouraging.

8.2. Achieved Benefits of the Research

8.2.1 Primary Benefits

Development of a novel agent-based adaptive join algorithm for the distributed data warehouse environment that will facilitate:

- The processing of geographically distributed data.
- The analysis of dynamic data warehouse data as opposed to snapshot or static data.
- The monitoring of its own processing to adapt accordingly to its environment at runtime.

8.2.2 Secondary benefits

- Build upon and extend software agent methodology to data warehouse design.
- Construct an agent-based framework for data warehouse and develop autonomous, adaptive and mobile agents for such systems.
- Gain a better understanding of the feasibility and effectiveness of using agent technology for adaptive query processing.

- Comprehension of impact of heterogeneous data schemas on the efficiency of query execution.
- Appreciation of strengths and limitations of adaptive join algorithms in distributed and dynamic environment.

8.3. Limitations

The work completed so far has provided a solid ground for further research in the area. An agent-based join algorithm called AJoin has been proposed, and an experimental study has been conducted to assess its feasibility and main characteristics. The outcome from the study has indicated that the basic model of the AJoin algorithm offers noticeable performance improvement over other modern join algorithms in the simulation environment modelled on a simplified distributed and dynamic data warehouse system.

In the experimental study, a controlled data warehousing experimental environment is used, in which the network speed and memory availability as the main factors affecting the comparison of join performance among the different types of join adaptive algorithm are identified and used. However, other factors in the real distributed data warehouse environments may also affect on join performance. For example, CPU power in multithread server environment could make the join performance more dynamic and difficult to compare amongst each other. It is worthwhile to evaluate the effectiveness of AJoin in a range of real world distributed data warehouse environments if possible.

Although we proposed an agent-based data warehousing architecture, it requires

considerable additional work to make the proposed architecture available which is beyond the scope of this research. As a result, the AJoin algorithm is to be implemented into a data warehouse query processing, because most of the current data warehousing systems are supported by commercial data base systems and have no API or facilities to allow a new join algorithm to add into their systems.

8.4. Further Research

The aim of this thesis has been to investigate the evolution of data warehouses architecture to identify architecture suitable for highly distributed data warehouses and the feasibility and effectiveness of utilising software agent technology to address some specific issues in data warehouses. This has successfully been achieved and an agent-based data warehouse architecture is proposed. During this work join operation as a key technique has been identified. A new agent-based join algorithm, AJoin, has been proposed to improve query processing performance in the dynamic and distributed data warehouse context. The main algorithm of AJoin was implemented and evaluated against other modern join algorithms. AJoin has exhibited better performance under distributed and dynamic data warehouse environments. The outcome of this research was very encouraging.

However, in order to create further research or commercial impact, AJoin need to be implemented in a real world data warehouse system. Ideally, a data warehouse system using the agent-based data warehousing architecture could be fully implemented; in which AJoin will be one of the join algorithms in its query processing unit.

A future extension for this work are identified in the five areas below; these are concerned with the provision of efficient and adaptive query processing, in particular agent-based join algorithms, for a dynamic and distributed data warehouse environment:

8.4.1 Heterogeneity in data schema

Data warehouses employ different data schemas compared to traditional database schemas used in OLTP system. A star schema is a popular choice for modelling data warehouses and data marts, and it is important that efficient mechanisms are available for the execution of queries modelled on the star schema (Weininger, 2002). Although the preliminary experimental outcome of AJoin algorithm has demonstrated its potential performance in a distributed and dynamic data warehouse environment, the experiment was based on a simplified data warehouse schema. It is important to further investigate the influence of heterogeneity of data schemas on the join algorithm.

8.4.2 Adaptive behaviour in query processing

Various query processing approaches could result in very different join performance. Unpredictability of server performance and network traffic make optimising query processing a challenging task (Khan, 2000). So far, the AJoin algorithm has exhibited remarkable adaptivity in join processing, which is mainly achieved by automated join method switching at runtime and its pipeline join feature. Queries in a data warehouse usually require some combination of selecting dimension tables, joining the dimension tables with fact tables, and some optional aggregation/summarisation functions. Therefore,

further exploration on adaptive behaviours in query processing is critical to the performance improvement of the join algorithm.

8.4.3 Effect of network environment

In a conventional network environment, transmission delay is regarded as the dominant factor in communication cost function. For that reason, many distributed query processing algorithms are devised to minimise the volume of data transmitted over the network. The basic model of the AJoin algorithm has been tested and shown to perform well in a lower-end speed network environment. However, technological advancement in high-speed networks such as fibre optical and ATM networks is increasingly making bandwidth-on-demand possible. It is necessary to establish a broader network simulation environment to enable a further investigation of the impact of various network environments on the join algorithm.

8.4.4 Impact of dynamic and distributed data warehouse environment

Data skew could result in a significant effect on join performance. So far, only small samples of data sets have been used for evaluating the join algorithm. In a dynamic and distributed data warehouse environment, it is essential for a join algorithm to deal with the problems resultant from data skew. Impact of dynamic and distributed data needs to be further examined in order to make the AJoin algorithm work more effectively.

8.4.5 Use of agent techniques to enhance adaptiveness and intelligence of join algorithms

Software agents as a promising approach is proposed in this work to enhance adaptiveness and intelligence of join algorithms for distributed systems. So far, the outcome from the preliminary evaluation of the basic model of AJoin together with its comparison with other modern join algorithms has been very encouraging. However, extending the basic framework and developing it into a comprehensive working algorithm remains the most substantial task for the next stage.

In summary, outcomes from the study have indicated that the basic model of the AJoin algorithm offers noticeable performance improvements over other modern join algorithms in the simulation environment modelled on a simplified distributed and dynamic data warehouse system. The further research will allow the algorithm to be extended to real networks with heterogeneous data schemas over dynamic and distributed data and create a considerable impact on distributed data warehousing systems.

8.5. Closing Remarks

This thesis has proposed an agent-based join algorithm called AJoin, which seamlessly integrates semi-join and ripple join techniques within a multi-agent system in order to improve join adaptability and reduce data transmission cost in a distributed data warehousing environment. Taking into account data warehouse features, AJoin aims to optimise join operations and achieve better join performance. The experimental

evaluation results demonstrate that AJoin consistently outperforms other adaptive join algorithms. In particular, AJoin exhibits the following advantages:

- Adaptiveness - able to adapt the join method at runtime to optimise join performance
- Mobility - able to utilise remote agents to filter unqualified tuples at the remote site, hence reducing data transmission cost
- Parallelism - able to utilise dynamic pipelined distributed parallel join to tackle unpredictable network conditions in distributed data warehouses
- Scalability - able to extend the join algorithm from 2-way join to n-way join

Agre, P. & Chapman, D. (1987) An implementation of a theory of activity. In: *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87).* pp.268-272.

Ahmad, A., Ahmad, M.S. & Yusoff, M.Z.. (2008) An exploratory review of software agents. In: *International Symposium on Information Technology, 2008. ITSim 2008.* IEEE, pp.1-8.

Anon (2002) Decoupled Query Optimization for Federated Database Systems. In: *Proceedings of the 18th International Conference on Data Engineering.* ICDE '02. IEEE Computer Society, p.716–.

Apers, P.M.G., Hevner, A.R. & Yao, S.B. (1983) Optimization Algorithms for Distributed Queries. *IEEE Transactions on Software Engineering*, SE-9 (1), pp.57 – 68.

Avnur, R. & Hellerstein, J.M. (2000) Eddies: Continuously Adaptive Query Processing. *IN SIGMOD*, p.p.261--272.

Babin, G. & Cheung, W. (2008) A Metadatabase-supported shell for distributed processing and systems integration. *Know.-Based Syst.*, 21 (7), p.pp.672–680.

Bell, D.A., Grimson, J.B. & Grimson, J. (1992) *Distributed database systems.* Addison-Wesley Pub. Co.

Bellifemine, F.L., Caire, G. & Greenwood, D. (2007) *Developing Multi-Agent Systems with JADE.* 1st ed. Wiley.

Bennett, T.A. & Bayrak, C. (2011) Bridging the data integration gap: from theory to implementation. *SIGSOFT Softw. Eng. Notes*, 36 (3), p.pp.1–8.

Bernstein, P.A. & Chiu, D.-ming W. (1981) Using semi-joins to solve relational queries. *JOURNAL OF THE ACM*, 28, p.p.25--40.

Berthold, H. & Meyer-Wegener, K. (2001) Schema Design and Query Processing in a Federated Multimedia Database System. In: *Proceedings of the 9th International Conference on Cooperative Information Systems.* CoopIS '01. Springer-Verlag, pp.285–300.

Bhashyam, R. (2004) Technology challenges in a data warehouse. In: *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30.* Toronto, Canada, VLDB Endowment, pp.1225-1226.

Bollacker, K.D., Lawrence, S. & Giles, C.L. (1998) An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications. *INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS*, p.p.116--123.

Brooks, R. (1986) A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2 (1), p.pp.14-23.

Brooks, R. (1991) Intelligence Without Representation. *ARTIFICIAL INTELLIGENCE*, 47, p.p.139--159.

Browning, D. & Mundy, J. (2001) Data Warehouse Design Considerations [Internet]. Available from: <http://msdn.microsoft.com/en-us/library/aa902672(v=sql.80).aspx> [Accessed 31 July 2011].

Cheung, W. & Hsu, C. (1996) The model-assisted global query system for multiple databases in distributed enterprises. *ACM Trans. Inf. Syst.*, 14 (4), p.pp.421–470.

Coronel, C., Morris, S. & Rob, P. (2009) *Database Systems: Design, Implementation, and Management*. 9th ed. Course Technology.

Dale, J. & DeRoure, D.C. (1997) Towards a Framework for Developing Mobile Agents for Managing Distributed Information Resources. *IN PROCEEDINS OF PRACTICAL APPLICATIONS OF INTELLIGENT AGENTS AND MULTI-AGENTS, PAAM'97.*

Deshpande, A., Deshp, A., Hellerstein, J.M. & Hellerstein, J.M. (2004) Lifting the Burden of History from Adaptive Query Processing. *IN VLDB*, p.p.948--959.

Ferguson, I.A. (1992) Touring Machines: autonomous agents with attitudes. *Computer*, 25 (5), p.pp.51-55.

Firestone, J.M. (1998) Architectural evolution in datawarehousing and distributed knowledge management architecture. *WHITE PAPER, EXECUTIVE INFORMATION SYSTEMS.*

Forrester, J. & Ledlie, J. (2002) *XJoin and the Benefits of Free Work.*

Franklin, S. (1997) *Artificial Minds (Bradford Book).* New edition. MIT Press.

Franklin, S. & Graesser, A. (1997) Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In: *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*. Springer-Verlag, pp.21–35.

Georgeff, M.P. & Rao, A.S. (1995) BDI agents: From theory to practice. , 95 (Technical Note 56), p.pp.312-319.

Goddard, W. & Melville, S. (2004) *Research Methodology: An Introduction.* Juta and Company Ltd.

Grosof, B. (1997) Building Commercial Agents: An IBM Research Perspective. , (RC 20835).

Haas, L.M., Carey, M.J., Livny, M. & Shukla, A. (1993) SEEKing the Truth about Ad Hoc Join Costs. *VLDB JOURNAL*, 6, p.p.241--256.

Haas, P.J. & Hellerstein, J.M. (1999) Ripple Joins for Online Aggregation. *Proceedings ACM SIGMOD International Conference on Management of Data*, pp.287–298.

Hackney, D. (1997) *Understanding and Implementing Successful Data Marts*. 1st ed. Addison Wesley Publishing Company.

Hasselbring, W. (2000) Research and practice in federated information systems. *SIGMOD Rec.*, 29 (4), pp.16–18.

Harris, E.P. & Ramamohanarao, K. (1996) Join algorithm costs revisited. *The VLDB Journal The International Journal on Very Large Data Bases*, 5, p.pp.64-84.

Hayes-Roth, B. (1995) An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72 (1-2), p.pp.329-365.

Honavar, V. & Dobbs, D. (2001) An Agent-Based Environment for Integrating and Analyzing Plant Genomic Databases. *In: PROCEEDINGS OF THE IEEE INFORMATION TECHNOLOGY CONFERENCE. SYRACUSE, NY.*

Honavar, V., Miller, L. & Wong, J. (1998) Distributed Knowledge Networks. *IN: PROCEEDINGS OF THE IEEE INFORMATION TECHNOLOGY CONFERENCE*, p.p.87--90.

Huhns, M.N. (1998) Agent Foundations for Cooperative Information Systems. *IN: PROC. S OF THE THIRD INTERNATIONAL CONFERENCE ON THE PRACTICAL APPLICATIONS OF INTELLIGENT AGENTS AND MULTI-AGENT TECHNOLOGY; LONDON 1998; EDITED BY H.S. NWANA AND D.T. NDUMU.*

Inmon, W.H. (1992) *Building the Data Warehouse*. 2nd Revised ed. Q E D Pub Co.

Inmon, W.H. (2005) *Building the Data Warehouse*. 4th ed. Wiley.

Inmon, W.H., Strauss, D. & Neushloss, G. (2008) *DW 2.0: The Architecture for the Next Generation of Data Warehousing*. Morgan Kaufmann.

Ives, Z., Florescu, D., Roquencourt, I., Friedman, M., Levy, A. & Weld, D. (1999) An Adaptive Query Execution System for Data Integration. , p.p.299--310.

Jean-Paul Arcangeli, A.H. (2004) Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing. *J. Database Manag.*, 15, pp.25–44.

Jennings, N.R. (2001) An agent-based approach for building complex software systems. *Commun. ACM*, 44 (4), p.pp.35–41.

Jennings, N.R. & Wooldridge, M.J. (1998) *Agent technology: foundations, applications, and markets*. Springer.

Jennings, N.R. & Wooldridge, M.J. (2010) *Agent Technology: Foundations, Applications, and Markets*. Springer.

Kang, H. & Roussopoulos, N. (1987) On Cost-effectiveness of a Semijoin in Distributed Query Processing. [Internet]. Available from: <http://drum.lib.umd.edu/handle/1903/4547> [Accessed 18 September 2011].

Kimball, R. & Ross, M. (2002) *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd ed. Wiley.

Kimball, R., Reeves, L., Ross, M. & Thornthwaite, W. (1998) *The Data Warehouse Lifecycle Toolkit : Expert Methods for Designing, Developing, and Deploying Data Warehouses*. Wiley.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, Joy & Becker, B. (2008) *The Data Warehouse Lifecycle Toolkit*. 2nd ed. John Wiley & Sons.

Lane, P. (2007) Oracle® Database Data Warehousing Guide [Internet].

Li, X., Gu, Y., Yue, D. & Yu, G. (2007) An Adaptive Join Strategy in Distributed Data Stream Management System. In: *2007 International Conference on Computational Intelligence and Security*. IEEE, pp.271-275.

Lieberman (1999) Let's browse: a collaborative browsing agent. *Knowledge-Based Systems*, 12 (8).

Litwin, W., Mark, L. & Roussopoulos, Nick (1990) Interoperability of multiple autonomous databases. *ACM Comput. Surv.*, 22 (3), p.pp.267–293.

Luo, G., Eiimann, C.J., Haas, P.J. & Naughton, J.F. (2002) A Scalable Hash Ripple Join Algorithm. *SIGMOD*, 2002, p.p.252--262.

Maes, P. (1995) Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38 (11), p.pp.108–114.

Moeller, R.A. (2000) *Distributed Data Warehousing Using Web Technology: How to Build a More Cost-Effective and Flexible Warehouse*. 1st ed. AMACOM.

Müller, J.P. (1996) *The Design of Intelligent Agents: A Layered Approach*. 1st ed. Springer.

MÜLLER, J.P. (1999) Architectures and Applications of Intelligent Agents: A Survey. *The Knowledge Engineering Review*, 13 (04), p.pp.353-380.

Mundy, Joy, Thornthwaite, W. & Kimball, R. (2011) *The Microsoft Data Warehouse Toolkit: With SQL Server 2008 R2 and the Microsoft Business Intelligence Toolset*. 2nd ed. John Wiley & Sons.

Nica, A. & Rundensteiner, Elke Angelika (1996) *The Dynamic Information Integration Model*.

Nwana, H.S. (1996) Software Agents: An Overview. *The Knowledge Engineering Review*, 11 (03), p.pp.205-244.

Olston, C. & Widom, J. (2005) Efficient Monitoring and Querying of Distributed, Dynamic Data via Approximate Replication. *IEEE Data Engineering Bulletin*, (special issue on In-Network Query Processing).

Parandoosh, F. (2007) Evaluating Agent-Oriented Software Engineering Methodologies. In: *2nd International Workshop on Soft Computing Applications, 2007. SOFA 2007*. IEEE, pp.169-174.

Qicheng Yu, McCann, J.A. & Fang Fang Cai (2009) An Agent-Based Adaptive Join Algorithm for Distributed Data Warehousing. In: *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:* pp.72-77.

Rahimi, S.K. & Haug, F.S. (2010) *Distributed Database Management Systems: A Practical Approach*. 1st ed. Wiley-IEEE Computer Society Pr.

Ramakrishnan, R. & Gehrke, J. (2002) *Database Management Systems*. 3rd ed. McGraw-Hill Science/Engineering/Math.

Rhodes, B. & Starner, T. (1996) Remembrance Agent: A continuously running automated information retrieval system. In: *Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM '96)*. pp.487-495.

Rundensteiner, E. A., Koeller, A. & Zhang, X. (2000) Maintaining data warehouses over changing information sources. *Commun. ACM*, 43 (6), pp.57–62.

Russell, S. & Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall.

Samos, J., Saltor, F., Sistac, J. & Bardés, A. (1998) Database Architecture for Data Warehousing: An Evolutionary Approach. In: *Proceedings of the 9th International Conference on Database and Expert Systems Applications*. Springer-Verlag, pp.746–756.

Sen, A. & Sinha, A.P. (2005) A comparison of data warehousing methodologies. *Commun. ACM*, 48 (3), pp.79–84.

Sheth, A.P. & Larson, J.A. (1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22 (3), pp.183–236.

Silberschatz, A., Korth, H. & Sudarshan, S. (2010) *Database System Concepts*. 6th ed. McGraw-Hill Science/Engineering/Math.

Smith, D., Cypher, A. & Spohrer, J. (1994) KidSim: programming agents without a programming language. *Communications of the ACM*, 37 (7), p.pp.54-67.

Theodoratos, D. & Sellis, T.K. (1999) Dynamic Data Warehouse Design. In: *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery*. DaWaK '99. London, UK, UK, Springer-Verlag, pp.1–10.

Urhan, T. & Franklin, M.J. (2000) XJoin: A Reactively-Scheduled Pipelined Join Operator. *IEEE DATA ENGINEERING BULLETIN*, 23, p.p.2000.

Vieira-Marques, P.M., Robles, S., Cucurull, J., Cruz-Correia, R.J., Navarro, G. & Marti, R. (2006) Secure Integration of Distributed Medical Data Using Mobile Agents. *IEEE Intelligent Systems*, 21 (6), p.pp.47-54.

vpnsp.com (2011) VPN Speed Test : Which VPN Service is the Fastest? [Internet]. Available from: <http://www.vpnsp.com/speed-test.html> [Accessed 7 November 2011].

Weiss, G. (1999) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press.

Widom, Jennifer (1995) Research problems in data warehousing. In: *Proceedings of the fourth international conference on Information and knowledge management*. CIKM '95. New York, NY, USA, ACM, pp.25–30.

Wooldridge, M. (2000) Agent-Oriented Software Engineering:The State of the Art. *AGENT-ORIENTED SOFTWARE ENGINEERING, VOLUME 1957 OF LECTURE NOTES IN COMPUTERS SCIENCE.*

Wooldridge, M. (2002) *An Introduction to MultiAgent Systems*. 1st ed. John Wiley & Sons.

Wooldridge, M. & Jennings, N. (1995) Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10 (2), p.pp.115-152.

Yang, J., Honavar, V., Miller, L. & Wong, J. (1998) Intelligent Mobile Agents for Information Retrieval and Knowledge Discovery from Distributed Data and Knowledge Sources. *IN PROC. OF THE IEEE INFORMATION TECHNOLOGY CONFERENCE.*

Zhong, N. (2001) *Intelligent agent technology: research and development*. World Scientific.